

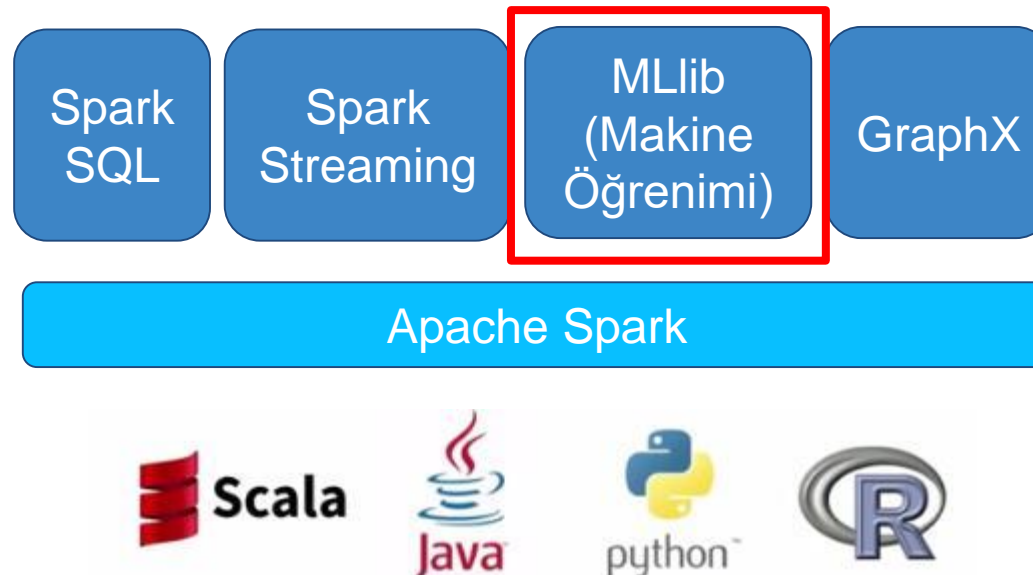


SPARK ile Makine Öğrenmesi

- **Spark'a Genel Bakış**
- **Spark MLlib'e Genel Bakış**
- **Spark MLlib Araçları**
- **Spark MLlib Avantajları**
- **Spark MLlib Veri Tipleri**
- **Spark MLlib İş Akışı**
- **Spark MLlib Algoritmaları**
 - **Sınıflandırma**
 - Doğrusal Destek Vektör Makinesi
 - Lojistik Regresyon
 - Karar Ağaçları
 - Rastgele Orman
 - Gradyan Artırılmış Ağaçlar
 - Naive Bayes

- **Regresyon**
 - Doğrusal Regresyon
 - Karar Ağaçları
 - Rastgele Orman
 - Gradyan Artırılmış Ağaçlar
- **Kümeleme**
 - K-Ortalama Kümeleme
 - Gauss Karışımı Kümeleme
 - Akan K-Ortalama Kümeleme
- **İşbirlikçi Filtreleme**
 - Alternatif En Küçük Kareler
- **Boyut İndirgeme**
 - Tekil Değer Ayrıştırma
 - Temel Bileşen Analizi

- Spark MLlib, Apache Spark'ın Makine Öğrenme bileşenidir.
- Spark'ın en önemli özelliklerinden biri, hesaplamayı büyük oranda **ölçeklendirme** yeteneğidir.
- Spark MLlib üzerinde, Spark çerçevesinde **ölçeklenebilir** ve özlu bir şekilde daha fazla **makine algoritması** uygulanabilir.



- Spark MLlib, Apache Spark'da makine öğrenimini yapmak için kullanılır. MLlib, popüler algoritmalar ve yardımcı programlardan oluşur.
- Spark makine öğrenmesi için kullanılabilecek iki paket vardır:

spark.mllib

- **RDD** ile çalışır
- Şuanda **bakım** modundadır

spark.ml

- ML pipeline'larını oluşturulması için **Dataframe** ile çalışır
- Spark'ın **birincil** Makine Öğrenme API'si

[Overview](#)[Programming Guides](#)[API Docs](#)[Deploying](#)[More](#)

MLlib: Main Guide

- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- Data types
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- Dimensionality reduction
- Feature extraction and transformation
- Frequent pattern mining
- Evaluation metrics
- PMML model export
- Optimization (developer)

Machine Learning Library (MLlib) Guide

MLlib is Spark's machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. At a high level, it provides tools such as:

- ML Algorithms: common learning algorithms such as classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction, transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
- Persistence: saving and load algorithms, models, and Pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Announcement: DataFrame-based API is primary API

The MLlib RDD-based API is now in maintenance mode.

As of Spark 2.0, the [RDD-based APIs](#) in the `spark.mllib` package have entered maintenance mode. The primary Machine Learning API for Spark is now the [DataFrame-based API](#) in the `spark.ml` package.

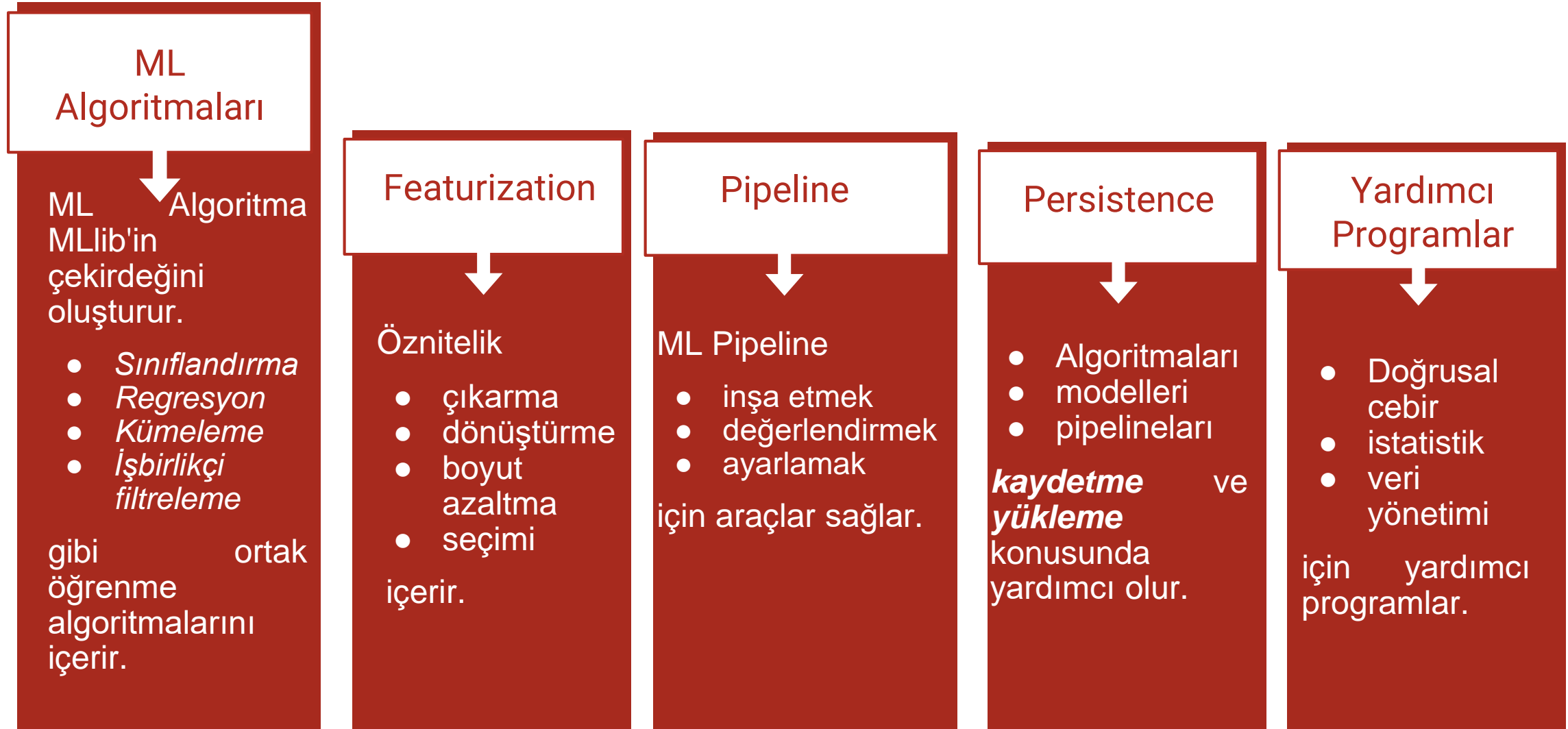
What are the implications?

- MLlib will still support the RDD-based API in `spark.mllib` with bug fixes.
- MLlib will not add new features to the RDD-based API.
- In the Spark 2.x releases, MLlib will add features to the DataFrames-based API to reach feature parity with the RDD-based API.
- After reaching feature parity (roughly estimated for Spark 2.2), the RDD-based API will be deprecated.
- The RDD-based API is expected to be removed in Spark 3.0.

Why is MLlib switching to the DataFrame-based API?

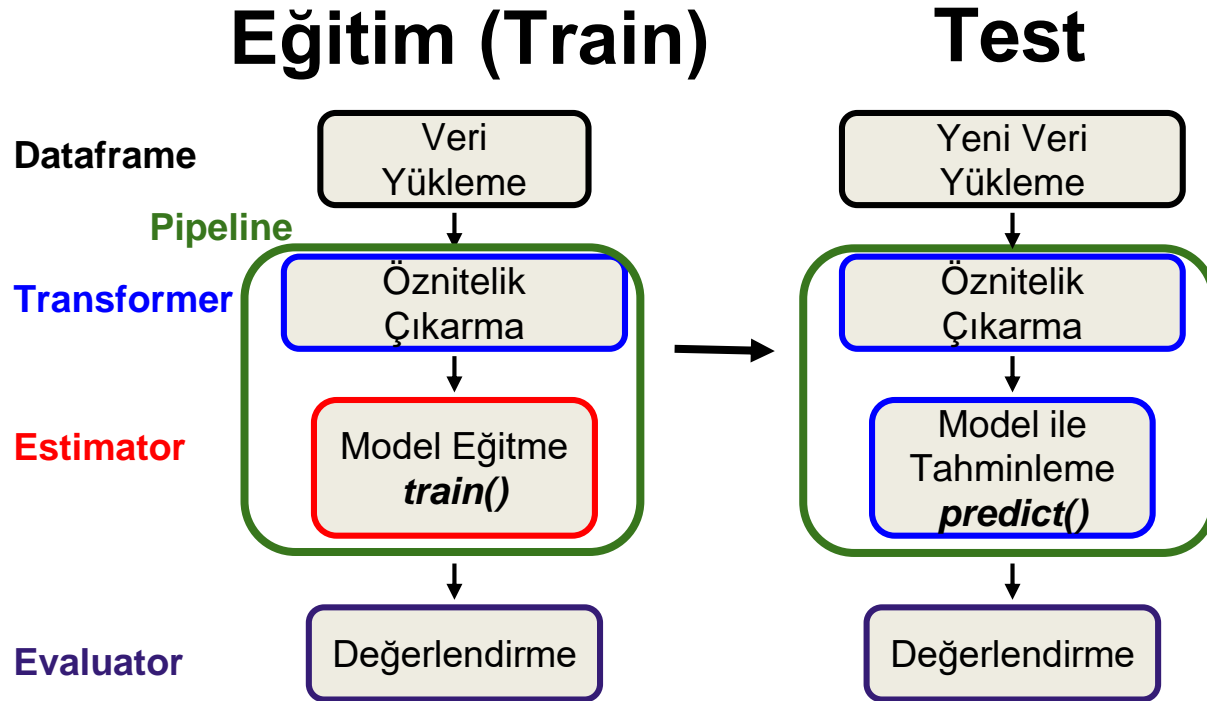
- DataFrames provide a more user-friendly API than RDDs. The many benefits of DataFrames include Spark Datasources, SQL/DataFrame queries, Tungsten and Catalyst optimizations, and uniform APIs across languages.
- The DataFrame-based API for MLlib provides a uniform API across ML algorithms and across multiple languages.
- DataFrames facilitate practical ML Pipelines, particularly feature transformations. See the [Pipelines guide](#) for details.

Spark MLlib aşağıdaki araçları sunar:



- Çoğu makine öğrenme kütüphanesi, **tek** bir makinede çalıştırılacak şekilde tasarlanmıştır
 - **Veri kümesinin boyutu**, kısmen bu makinedeki yerel hafızaya sığabilecek kadardır.
 - **Verilerin işlenmesi** sadece bir makinenin CPU kaynakları üzerinde gerçekleşir.
- Spark onlarca, yüzlerce veya binlerce makinenin aynı anda **paralel** işlemler yürütebilir.
- İki Spark paketi arasında **ML**'in **MLlib**'e göre birçok avantajı vardır.
 - ML, DataFrame'lerde çalıştığı için, temel Catalyst optimizasyon motoru sayesinde daha fazla **esneklik ve otomatik performans geliştirme** sağlar.

Yerel (Local) Vektörler	<ul style="list-style-type: none">• Seyrek (Sparse) Vektör• Yoğun (Dense) Vektör• <code>mllib.linalg.Vectors</code>
Labeled Point	<ul style="list-style-type: none">• Güdümlü (Supervised) Öğrenme için kullanılır• Öznitelik (feature) vektörü + etiketi (label)
Matris	<ul style="list-style-type: none">• Yerel (Local) Matris<ul style="list-style-type: none">○ Seyrek (Sparse) Matris○ Yoğun (Dense) Matris• Dağıtık (Distiributed) Matris<ul style="list-style-type: none">○ RowMatrix○ IndexedRowMatrix○ CoordinateMatrix○ BlockMatrix
Çeşitli Modeller	<ul style="list-style-type: none">• Modeller, <code>train()</code> yönteminden kaynaklanmaktadır• Yeni bir veri noktasına veya noktaların RDD'sine uygulanacak Model'in <code>predict()</code> yöntemi



DataFrame: ML paketi, çeşitli veri türlerini tutabilen DataFrame'i kullanır. Örneğin, bir DataFrame metni, özellik vektörlerini, gerçek etiketleri ve tahminleri depolayan farklı sütunlara sahip olabilir.

Transformer: Bir Transformer bir DataFrame'i başka bir DataFrame'e dönüştürebilen bir algorithmadır. Örneğin, bir ML modeli, özellikleri olan bir DataFrame'i tahminlerle bir DataFrame'e dönüştüren bir Transformatördür. *transform()*

Estimator: Bir Estimator, bir Transformer üretmek için bir DataFrame'e sığabilen bir algoritma. Örneğin bir öğrenme algoritması, bir DataFrame üzerinde eğitilen ve bir model üreten bir Estimatordür. *fit()*

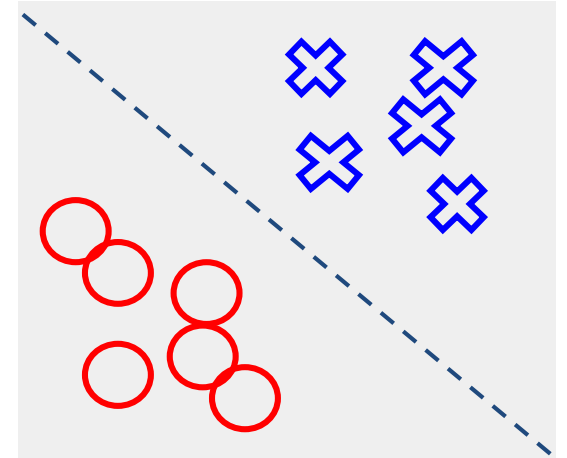
Pipeline: Bir Pipeline, bir ML iş akışını belirtmek için birlikte birçok Transformer ve Estimatorü zincirler.

- **Sınıflandırma (Classification)**
 - **Doğrusal Destek Vektör Makinesi** (Linear Support Vector Machine)
 - **Lojistik Regresyon** (Logistic regression)
 - **Naive Bayes**
 - **Karar Ağaçları** (Decision Trees)
 - **Ağaçlar Topluluğu** (Ensemble of Trees)
 - Rastgele Orman (Random Forest)
 - Gradyan Artırılmış Ağaçlar (Gradient Boost Trees)
- **Regresyon (Regression)**
 - **Doğrusal Regresyon** (Linear regression)
 - **Ridge regresyon** (Ridge regression): **RidgeRegressionWithSGD** sınıfı uygulanan, L2 düzenlemesiyle doğrusal en küçük kareler
 - **Lasso**: **LassoWithSGD** sınıfı uygulanan, L1 düzenlemesiyle doğrusal en küçük kareler
 - **İzotonik regresyon** (Isotonik)

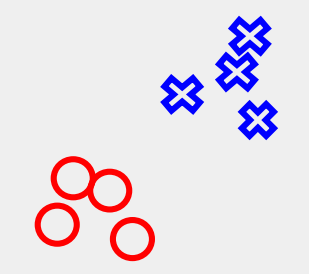
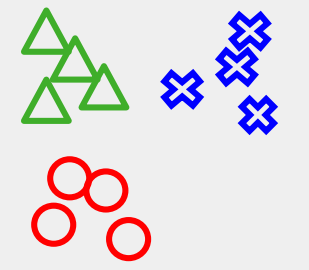
- **Kümeleme (Clustering)**
 - K- Ortalama (K-Means)
 - Gauss karışımı (Gaussian mixture)
 - Gizli Dirichlet tahsisi (LDA: Latent Dirichlet Allocation)
 - Akan k-ortalama (Streaming k-means)
- **İşbirlikçi Filtreleme (Collaborative Filtering)**
 - Alternatif en küçük kareler (Alternating least squares):
- **Boyut İndirgeme (Dimensionality Reduction)**
 - Tekil Değer Ayırıştırma (SVD: Singular Value Decomposition)
 - Temel Bileşen Analizi (PCA: Principal Component Analysis)

Sınıflandırma

- Doğrusal Destek Vektör Makinesi
- Lojistik Regresyon
- Karar Ağaçları
- Rastgele Orman
- Gradyan Artırılmış Ağaçlar
- Naive Bayes



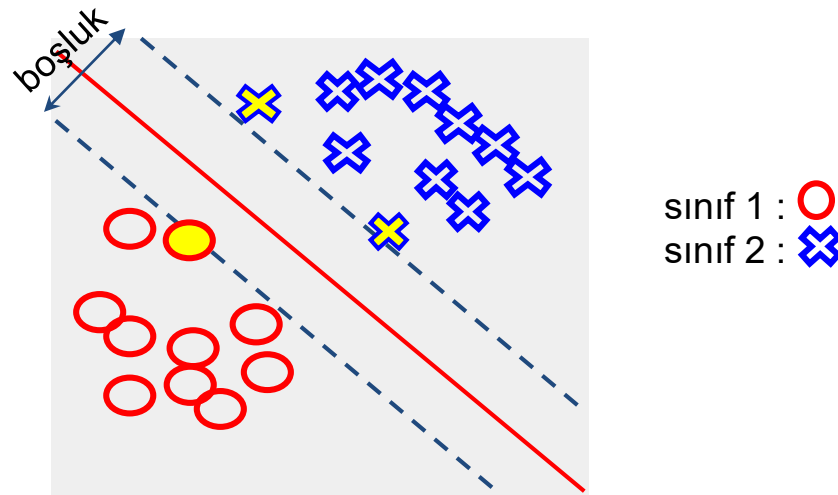
- **Sınıflandırma**, öğeleri kategorilere ayırmayı amaçlar. En yaygın sınıflandırma türü **ikili sınıflandırma**dır; burada genelde pozitif ve negatif olarak adlandırılan iki kategori bulunur. Eğer ikiden fazla kategori varsa, **çok sınıflı sınıflandırma** denir.
- **spark.mllib** paketi, **ikili(binary) sınıflandırma ve çok sınıflı (multiclass) sınıflandırma** için çeşitli yöntemleri desteklemektedir. Aşağıdaki tablo, her bir sorun türünün desteklediği algoritmaları özetlemektedir.

Problem tipi	Şekil	Desteklenen Yöntem
İkili sınıflandırma		lineer SVM, lojistik regresyon, karar ağaçları, rastgele ormanlar, gradyan artırılmış ağaçlar, naive bayes
Çok sınıflı sınıflandırma		lojistik regresyon, karar ağaçları, rastgele ormanlar, naive bayes

sınıf 1 : 
sınıf 2 : 
sınıf 3 : 

Doğrusal Destek Vektör Makinesi :

- Büyük ölçekli sınıflandırma görevleri için standart bir yöntemdir.
- Eğitim verilerindeki herhangi bir noktadan en uzak olan iki sınıf arasında bir karar sınırı bulan vektör uzayı tabanlı makine öğrenme yöntemi olarak tanımlanabilir.



```
from pyspark.ml.classification import LinearSVC
```

```
# Eğitim verisini yükleyin
```

```
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")
```

```
lsvc = LinearSVC(maxIter=10, regParam=0.1)
```

```
# Modeli uygulayın
```

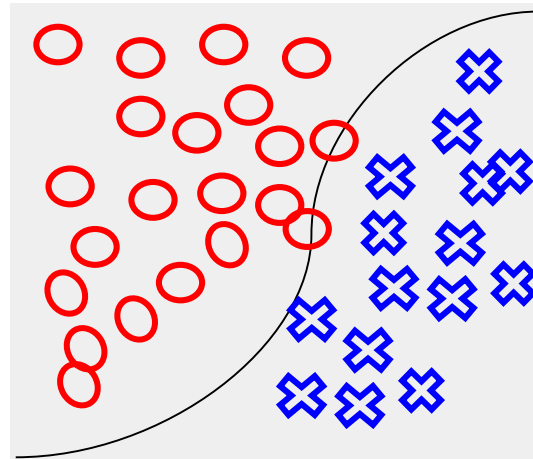
```
lsvcModel = lsvc.fit(training)
```

```
# Katsayıları ve intercepti yazdırın
```

```
print("Coefficients: " + str(lsvcModel.coefficients))
```

```
print("Intercept: " + str(lsvcModel.intercept))
```

- İstatistikte, **lojistik regresyon** bağımlı değişkenin kategorik olduğu bir regresyon modelidir.
- İkili bağımlı bir değişkeni kapsar:
 - başarılı / başarısız, kazan / kaybet, canlı / ölü veya sağlıklı / hasta gibi sonuçları temsil eden "0" ve "1".
- Bağımlı değişkenin ikiden fazla çıktı kategorisine sahip olduğu durumlarda, **multinomial lojistik regresyonda** analiz edilebilir.



sınıf 1 : ○
sınıf 2 : ×

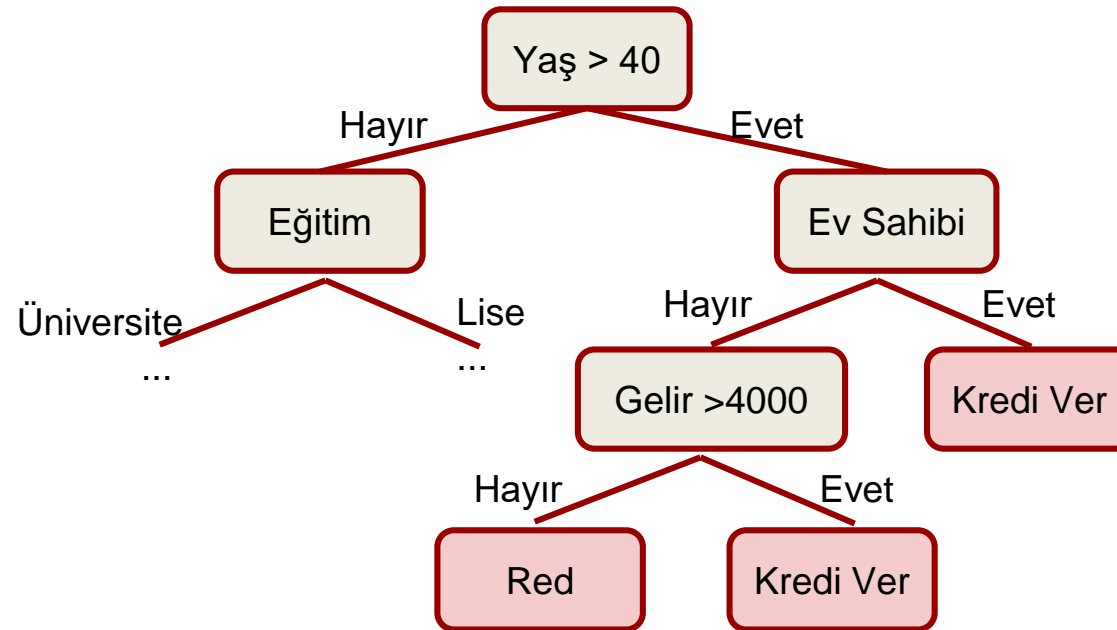

```
from pyspark.ml.classification import LogisticRegression
# Eğitim verisini yükleyin
training =
spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Modeli uygulayın
lrModel = lr.fit(training)

# Katsayıları ve intercepti yazdırın
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

- **Karar ağaçları ve toplulukları**, *sınıflandırma* ve *regresyonun* makine öğrenme görevleri için popüler yöntemlerdir.
- Karar ağaçları, **yorumlanması**, **kategorik özellikleri** ele alması, çok katmanlı **sınıflandırma ayarını genişletmesi**, özellik ölçeklendirmesi gerektirmemesi ve doğrusal olmayanlıkları ve özellik etkileşimlerini yakalayabilmeleri nedeniyle yaygın olarak kullanılmaktadır.



```
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer

# Veri kümesini LibSVM biçiminde okuyup DataFrame olarak yükleyin.
data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

# Veriye etiket sütunu ekleyin.

labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
# Kategorik özellikleri otomatik olarak tanımlayın ve indeks oluşturun.
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures",
maxCategories=4).fit(data)

# Veriyi eğitim ve test için ayırın.

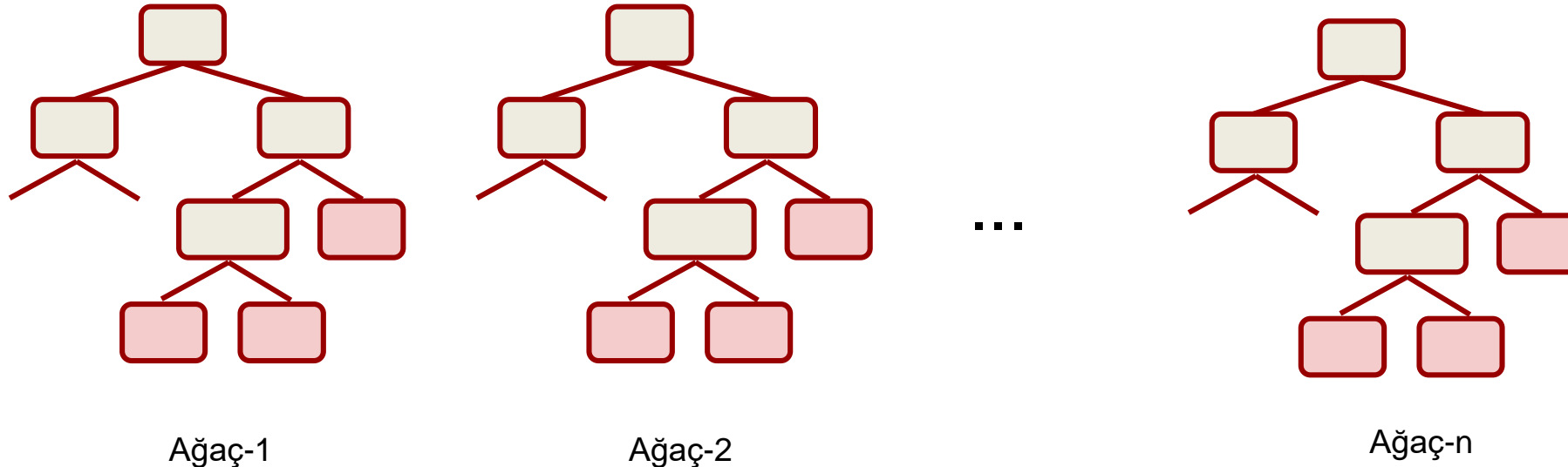
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Karar ağacı modelini eğitin.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Zincir dizinleyicileri ve karar ağacı pipelineda
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Eğitim modeli. Bu da indeksleri çalıştırır.
model = pipeline.fit(trainingData)
```

- **Rasgele ormanlar**, karar ağaçlarının topluluklarıdır.
- Rasgele ormanlar, **sınıflandırma ve regresyon** için en başarılı makine öğrenme modellerinden biridir.
- Aşırı uyum riskini azaltmak için **birçok karar ağacı** bir araya getirirler.
- Karar ağaçları gibi, rastgele ormanlar kategorik özellikleri idare eder, çok sınıflı sınıflandırma ayarını genişletir, özellik ölçeklendirmeye ihtiyaç duymaz ve doğrusal olmayan ve özellik etkileşimlerini yakalayabilir.



```
from pyspark.ml import Pipeline
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer,
VectorIndexer

# Veriyi yükleyin, ayrıştırın ve DataFrame dönüştürün
data =
spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

# Veriye etiket sütunu ekleyin.
labelIndexer = StringIndexer(inputCol="label",
outputCol="indexedLabel").fit(data)

# Kategorik özellikleri otomatik olarak tanımlayın ve indeks
oluşturun.
featureIndexer = VectorIndexer(inputCol="features",
outputCol="indexedFeatures", maxCategories=4).fit(data)
```

```
# Veriyi eğitim ve test için ayırın.
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Rastgele Orman modelini eğitin.
rf = RandomForestClassifier(labelCol="indexedLabel",
featuresCol="indexedFeatures", numTrees=10)

# İndeks eklenmiş etiketleri orijinal etiketlere
dönüştür.
labelConverter = IndexToString(inputCol="prediction",
outputCol="predictedLabel", labels=labelIndexer.labels)

# Zincir dizinleyicileri ve karar ağacı pipelineda
pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
rf, labelConverter])

# Eğitim modeli. Bu da indeksleri çalıştırır.
model = pipeline.fit(trainingData)
```

- **Gradyan Artırılmış Ağaçlar (GBT)** karar ağaçlarının topluluklarıdır.
- GBT'ler, bir kayıp fonksiyonunu en aza indirmek için karar ağaçlarını tekrarlar.
- Karar ağaçları gibi, GBTler kategorik özellikleri ele alır, çok sınıflı sınıflandırma ayarını genişletir, özellik ölçeklendirmeye ihtiyaç duymazlar ve doğrusal olmayan ve özellik etkileşimlerini yakalarlar.

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import GBClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
```

Veriyi yükleyin, ayrıştırın ve DataFrame dönüştürün.

```
data =
    spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")
```

Veriye etiket sütunu ekleyin.

```
labelIndexer = StringIndexer(inputCol="label",
    outputCol="indexedLabel").fit(data)
```

Kategorik özellikleri otomatik olarak tanımlayın ve indeks oluşturun

```
featureIndexer = VectorIndexer(inputCol="features",
    outputCol="indexedFeatures", maxCategories=4).fit(data)
```

Veriyi eğitim ve test için ayırın.

```
(trainingData, testData) = data.randomSplit([0.7, 0.3])
```

GBT modelini eğitin.

```
gbt = GBClassifier(labelCol="indexedLabel",
    featuresCol="indexedFeatures", maxIter=10)
```

Zincir dizinleyicileri ve karar ağacı pipelineda

```
pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
    gbt])
```

Eğitim modeli. Bu da indeksleri çalıştırır.

```
model = pipeline.fit(trainingData)
```

- **Naive Bayes**, özelliklerin her çifti arasında **bağımsızlık varsayımı** ile basit bir çok sınıflı sınıflandırma algoritmasıdır.
- Eğitim verisine tek bir geçişte, verilen her bir özelliğin **koşullu olasılık dağılımını** hesaplar ve ardından bir gözlem verilen koşullu olasılık dağılımını hesaplamak için **Bayes teoremi** uygular ve bunu tahmin için kullanır.


```
from pyspark.ml.classification import NaiveBayes
# Eğitim verisini yükleyin
data =
spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.tx
t")
# Veriyi eğitim ve test için ayırın.
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

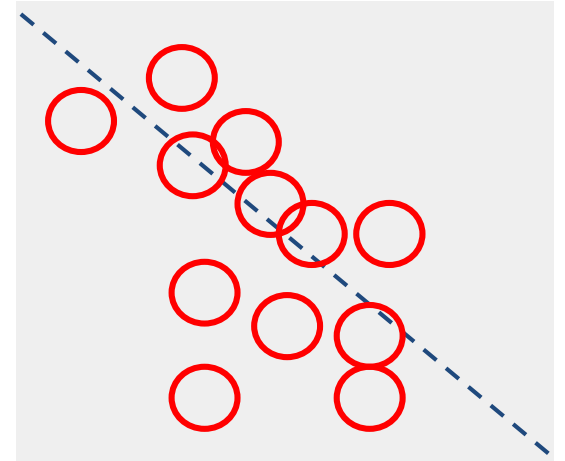
# Eğiticiyi yarat ve parametrelerini ayarla
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# Modeli eğit
model = nb.fit(train)
```

LAB: Sınıflandırma algoritmalarına örnekler

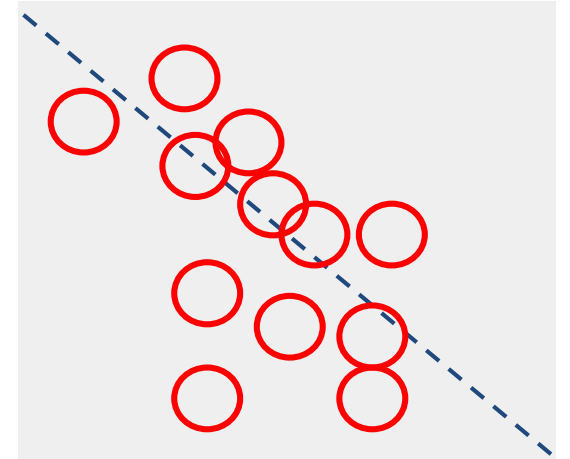
Regresyon

- Doğrusal Regresyon
- Karar Ağaçları
- Rastgele Orman
- Gradyan Artırılmış Ağaçlar



- **Regresyon analizi**, iki ya da daha çok değişken arasındaki ilişkiyi ölçmek için kullanılan analiz metodudur. Eğer tek bir değişken kullanılarak analiz yapılıyorsa buna **tek değişkenli regresyon**, birden çok değişken kullanılıyorsa **çok değişkenli regresyon** analizi olarak isimlendirilir.
- **spark.mllib** paketi, **regresyon analizi** için çeşitli yöntemleri desteklemektedir. Aşağıdaki tablo, her bir sorun türünün desteklediği algoritmaları özetlemektedir.

Problem tipi	Desteklenen Yöntem
Regresyon	<ul style="list-style-type: none">• Doğrusal en küçük kareler• Lasso• Ridge regresyon• Karar ağaçları• Rastgele ormanlar• Gradyan artırılmış ağaçlar• İzotonik regresyon



```
from pyspark.ml.regression import LinearRegression

# Veriyi yükleyin
training = spark.read.format("libsvm")\
    .load("data/mllib/sample_linear_regression_data.txt")

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Model eğitilir
lrModel = lr.fit(training)

# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))
```

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
# Veriyi yükleyin.
data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

# Kategorik özellikleri otomatik olarak tanımlayın ve indeks oluşturun.
featureIndexer = VectorIndexer(inputCol="features",
outputCol="indexedFeatures",      maxCategories=4).fit(data)

# Veri eğitim ve test olarak bölünür
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Karar Ağacı modeli eğitilir
dt = DecisionTreeRegressor(featuresCol="indexedFeatures")
pipeline = Pipeline(stages=[featureIndexer, dt])
model = pipeline.fit(trainingData)
```

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
```

```
# Load and parse the data file, converting it to a DataFrame.
data =
    spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")
```

```
# Automatically identify categorical features, and index them.
```

```
# Set maxCategories so features with > 4 distinct values are
    treated as continuous.
```

```
featureIndexer = \
    VectorIndexer(inputCol="features",
        outputCol="indexedFeatures", maxCategories=4).fit(data)
```

```
# Split the data into training and test sets (30% held
    out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])
```

```
# Train a RandomForest model.
rf = RandomForestRegressor(featuresCol="indexedFeatures")
```

```
# Chain indexer and forest in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, rf])
```

```
# Train model. This also runs the indexer.
model = pipeline.fit(trainingData)
```

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import GBRegressor
from pyspark.ml.feature import VectorIndexer

# Load and parse the data file, converting it to a DataFrame.
data =
    spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

# Automatically identify categorical features, and index them.
# Set maxCategories so features with > 4 distinct values are
# treated as continuous.
featureIndexer = \
    VectorIndexer(inputCol="features",
        outputCol="indexedFeatures", maxCategories=4).fit(data)
```

```
# Split the data into training and test sets (30% held
# out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a GBT model.
gbt = GBRegressor(featuresCol="indexedFeatures",
    maxIter=10)

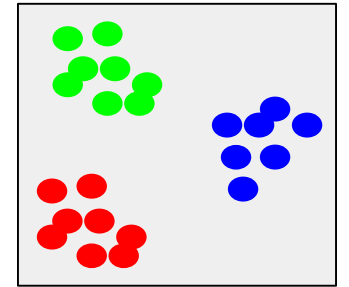
# Chain indexer and GBT in a Pipeline
pipeline = Pipeline(stages=[featureIndexer, gbt])

# Train model. This also runs the indexer.
model = pipeline.fit(trainingData)
```


LAB: Regresyon algoritmalarına örnekler

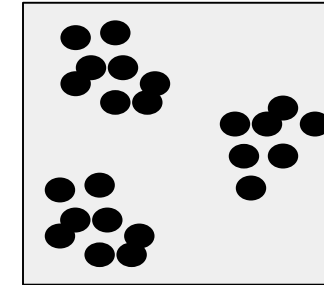
Kümeleme

- K- Ortalama (K-Means)
- Gauss karışımı (Gaussian mixture)
- Akan k-ortalama (Streaming k-means)

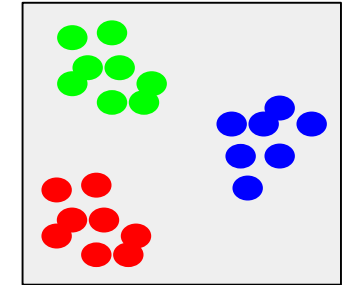


- **Kümeleme**, güdümsüz (unsupervised) bir öğrenme problemidir.
- Örneklerin alt gruplarını, benzerlik kavramına dayanarak birbirleriyle gruplamayı hedeflemektedir.

Problem tipi	Desteklenen Yöntem
Kümeleme	<ul style="list-style-type: none">• K- Ortalama (K-Means)• Gauss karışımı (Gaussian mixture)• Gizli Dirichlet tahsisi (LDA: Latent Dirichlet Allocation)• Akan k-ortalama (Streaming k-means)

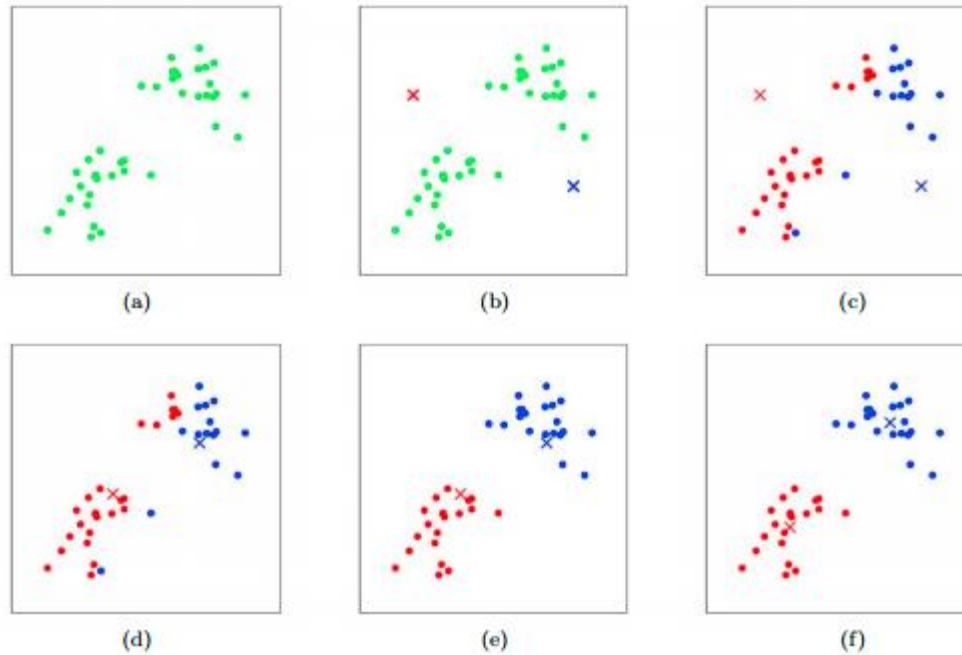


Kümelenmemiş
veri



Kümelenmiş
veri

- **K-ortalama**, veri noktalarını önceden tanımlanmış sayıda (k) küme olarak kümelendiren en yaygın kullanılan kümeleme algoritmalarından biridir.



```
from pyspark.ml.clustering import KMeans
# Veriyi yükleyin
dataset =
spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.
txt")

# k-ortalama modelini eğitin
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Sonuçları görüntüleyin
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

- **Gauss Karışımı Modeli**, noktaların her biri kendi olasılığı olan k Gauss alt dağılımlarından birinden alındığı bileşik bir dağılımı temsil eder.

```
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)
```

İşbirlikçi Filtreleme

- Alternatif En Küçük Kareler

- İşbirlikçi filtreleme, **öneri sistemlerinde** yaygın olarak kullanılır. Bu teknikler, bir kullanıcı-öge ilişkilendirme matrisinin eksik girdilerini doldurmayı amaçlamaktadır.
- **spark.mllib** şu anda, kullanıcıların ve ürünlerin, eksik girdileri tahmin etmek için kullanılabilecek küçük bir gizli(latent) faktör setiyle tanımlandığı model tabanlı ortak filtrelemeyi desteklemektedir.
- **spark.mllib**, bu gizli faktörleri öğrenmek için **alternatif en küçük kareler (ALS) algoritmasını** kullanır.

	Salih			
		4	5	5
	Fatma		5	5
	Okan		5	?

- ALS, K boyutunda seyrek kullanıcı madde derecelendirme matrisini, iki yoğun matrisin, $U \times K$ ve $I \times K$ boyutlarındaki
- Kullanıcı ve Öğe faktör matrislerinin ürünü olarak yaklaştırmaktadır (aşağıdaki resme bakınız). Faktör matrislerine gizli özellik modelleri denir.
- Faktör matrisleri, algoritmanın keşfetmeye çalıştığı gizli özellikleri temsil eder.
- Bir matris, her bir kullanıcının gizli veya gizlenmiş özelliklerini tanımlamaya çalışır ve her filmin gizli özelliklerini tanımlamaya çalışır.

```
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
lines =
spark.read.text("data/mllib/als/sample_movielens_ratings.txt").rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]),movieId=int(p[1]),
                                rating=float(p[2]), timestamp=long(p[3])))
ratings = spark.createDataFrame(ratingsRDD)

# Veri eğitim ve test için bölünür
(training, test) = ratings.randomSplit([0.8, 0.2])

# Model eğitilir
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId",
ratingCol="rating")
model = als.fit(training)
```

Boyut İndirgeme

- Tekil Değer Ayrıştırma (SVD: Singular Value Decompositon)
- Temel Bileşen Analizi (PCA: Principal Component Analysis)

- Boyutsal boyut azaltma, dikkate alınan değişken sayısını azaltma işlemidir.
- Ham ve gürültülü özelliklerden gizli özellikler çıkarmak veya yapıyı korurken verileri sıkıştırmak için kullanılabilir.
- spark.mllib, RowMatrix sınıfında boyut azalması için destek sağlar.

Problem tipi	Desteklenen Yöntem
Boyut İndirgeme	<ul style="list-style-type: none">• Temel Bileşen Analizi (PCA: Principal Component Analysis)

- **Temel bileşen analizi (PCA)**, ilk koordinatın mümkün olan en büyük varyansa sahip olması ve sonuçta elde edilen her koordinatın mümkün olan en büyük varyansa sahip olacağı şekilde bir dönme bulmak için istatistiksel bir yöntemdir.
- Dönme matrisinin sütunlarına ana bileşenler denir. PCA, boyutsal azaltmada yaygın olarak kullanılır.

```
from pyspark.ml.feature import PCA
from pyspark.ml.linalg import Vectors

data = [(Vectors.sparse(5, [(1, 1.0), (3, 7.0)]),),
        (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
        (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
df = spark.createDataFrame(data, ["features"])

pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")
model = pca.fit(df)

result = model.transform(df).select("pcaFeatures")
result.show(truncate=False)
```

Teşekkür Ederiz..

www.b3lab.org



TÜBİTAK BİLGEM Gebze Yerleşkesi, Gebze, Kocaeli, TÜRKİYE, 41470
T: +90 262 675 23 92 E: b3lab.iletisim@tubitak.gov.tr