Week-9

1. Logistic Regeression
2. Decision Tree
3. Naïve Bayes

from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.classification import NaiveBayes

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Step 1: Start a Spark session

spark = SparkSession.builder \

.appName("NaiveBayesExample") \

.getOrCreate()

# Step 2: Create a simple dataset

data = [

(0.0, 1.0, 3.0, 0.0), # (feature1, feature2, feature3, label)

(1.0, 2.0, 4.0, 1.0),

(0.0, 1.5, 3.5, 0.0),

(1.0, 3.0, 5.0, 1.0),

(0.0, 2.5, 4.5, 0.0),

(1.0, 3.5, 6.0, 1.0)

]

columns = ["feature1", "feature2", "feature3", "label"]

df = spark.createDataFrame(data, columns)

# Step 3: Combine features into a single 'features' column

assembler = VectorAssembler(inputCols=["feature1", "feature2", "feature3"], outputCol="features")

df = assembler.transform(df).select("features", "label")

# Step 4: Split the data into training and testing datasets

train\_data, test\_data = df.randomSplit([0.8, 0.2], seed=42)

# Step 5: Initialize and train the Naive Bayes classifier

nb = NaiveBayes(featuresCol="features", labelCol="label", modelType="multinomial")

nb\_model = nb.fit(train\_data)

# Step 6: Make predictions

predictions = nb\_model.transform(test\_data)

# Step 7: Evaluate the model

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")

accuracy = evaluator.evaluate(predictions)

print(f"Accuracy: {accuracy:.2f}")

# Step 8: Display predictions

predictions.select("features", "label", "prediction", "probability").show()

# Stop the Spark session

spark.stop()

WEEK-10

k-Mean in Pyspark

from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

# Step 1: Start Spark session

spark = SparkSession.builder \

.appName("KMeansExample") \

.getOrCreate()

# Step 2: Create a simple dataset

data = [

(1.0, 1.0),

(1.5, 1.5),

(3.0, 3.0),

(5.0, 5.0),

(3.5, 3.5),

(4.5, 4.5),

(3.0, 4.0)

]

columns = ["feature1", "feature2"]

df = spark.createDataFrame(data, columns)

# Step 3: Combine features into a single 'features' column

assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")

df = assembler.transform(df).select("features")

# Step 4: Apply K-Means Clustering

kmeans = KMeans(featuresCol="features", k=2, seed=42) # k=2 means 2 clusters

model = kmeans.fit(df)

# Step 5: Make predictions

predictions = model.transform(df)

# Step 6: Evaluate the model

evaluator = ClusteringEvaluator(featuresCol="features", metricName="silhouette", distanceMeasure="squaredEuclidean")

silhouette = evaluator.evaluate(predictions)

print(f"Silhouette with squared Euclidean distance: {silhouette:.2f}")

# Step 7: Display cluster centers

centers = model.clusterCenters()

print("Cluster Centers:")

for center in centers:

print(center)

# Step 8: Show predictions

predictions.show()

# Stop Spark session

spark.stop()

Gaussian Mixture Model(GMM)

from pyspark.sql import SparkSession

from pyspark.ml.clustering import GaussianMixture

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.linalg import Vectors

# Step 1: Create a Spark session

spark = SparkSession.builder.appName("GaussianMixtureExample").getOrCreate()

# Step 2: Create a sample dataset

data = [

(0.0, 0.1),

(1.0, 0.9),

(1.1, 0.8),

(8.0, 8.0),

(9.0, 8.0),

(8.5, 8.5)

]

columns = ["x", "y"]

df = spark.createDataFrame(data, columns)

# Step 3: Assemble features into a vector

assembler = VectorAssembler(inputCols=["x", "y"], outputCol="features")

vectorized\_data = assembler.transform(df)

# Step 4: Create and configure GaussianMixture model

gmm = GaussianMixture(featuresCol="features", k=2) # k is the number of clusters

# Step 5: Fit the model

model = gmm.fit(vectorized\_data)

# Step 6: Make predictions

predictions = model.transform(vectorized\_data)

predictions.show()

# Step 7: Analyze the results

# Get the Gaussian components (mean and covariance of each cluster)

for i, gaussian in enumerate(model.gaussiansDF.collect()):

print(f"Gaussian {i}: Mean = {gaussian['mean']}, Covariance = {gaussian['cov']}")

Week 11

Collaborative filtering using Spark ML

from pyspark.sql import SparkSession

from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator

# Step 1: Create a Spark session

spark = SparkSession.builder.appName("CollaborativeFilteringExample").getOrCreate()

# Step 2: Prepare the data (example data with userId, itemId, and rating)

data = [

(0, 0, 5.0),

(0, 1, 3.0),

(0, 2, 4.0),

(1, 0, 4.0),

(1, 1, 4.0),

(1, 2, 2.0),

(2, 0, 1.0),

(2, 1, 5.0),

(2, 2, 4.0)

]

columns = ["userId", "itemId", "rating"]

df = spark.createDataFrame(data, columns)

# Step 3: Split the data into training and test sets

train, test = df.randomSplit([0.8, 0.2])

# Step 4: Create the ALS model

als = ALS(maxIter=10, regParam=0.1, userCol="userId", itemCol="itemId", ratingCol="rating", coldStartStrategy="drop")

# Step 5: Fit the model to the training data

model = als.fit(train)

# Step 6: Make predictions

predictions = model.transform(test)

# Show some predictions

predictions.show()

# Step 7: Evaluate the model using RMSE

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")

rmse = evaluator.evaluate(predictions)

print(f"Root-mean-square error = {rmse}")

# Step 8: Generate top 5 recommendations for each user

userRecs = model.recommendForAllUsers(5)

userRecs.show()

# Step 9: Generate top 5 recommendations for each item

itemRecs = model.recommendForAllItems(5)

itemRecs.show()

# Stop the Spark session

spark.stop()

Week-12

Implement FP-Growth using Spark ML Library

from pyspark.sql import SparkSession

from pyspark.ml.fpm import FPGrowth

from pyspark.sql.functions import col

# Step 1: Create a Spark session

spark = SparkSession.builder.appName("FPGrowthExample").getOrCreate()

# Step 2: Prepare the data (example transactions)

data = [

(0, ["milk", "bread", "butter"]),

(1, ["milk", "bread"]),

(2, ["milk", "butter"]),

(3, ["bread", "butter"]),

(4, ["milk", "bread", "butter", "jam"]),

(5, ["milk", "bread", "jam"])

]

columns = ["id", "items"]

df = spark.createDataFrame(data, columns)

# Step 3: Create the FP-Growth model

fpgrowth = FPGrowth(itemsCol="items", minSupport=0.3, minConfidence=0.7)

# Step 4: Fit the model

model = fpgrowth.fit(df)

# Step 5: Get frequent itemsets

frequent\_itemsets = model.freqItemsets

frequent\_itemsets.show()

# Step 6: Get association rules

association\_rules = model.associationRules

association\_rules.show()

# Step 7: Generate predictions (find frequent itemsets for new transactions)

new\_data = [

(6, ["milk", "bread"]),

(7, ["butter", "jam"]),

]

new\_df = spark.createDataFrame(new\_data, columns)

predictions = model.transform(new\_df)

predictions.show()

# Step 8: Stop the Spark session

spark.stop()