[PySpark Machine Learning (MLlib)] [cheatsheet]

1. Data Preparation

- Create a dense vector: from pyspark.ml.linalg import Vectors; dense_vec = Vectors.dense([1.0, 2.0, 3.0])
- Create a sparse vector: sparse_vec = Vectors.sparse(5, [1, 3], [2.0, 4.0])
- Create a labeled point: from pyspark.ml.feature import LabeledPoint; labeled_point = LabeledPoint(1.0, dense_vec)
- Create a dataset from an RDD: dataset = spark.createDataFrame(rdd, schema)

2. Feature Transformation

- Binarizer: from pyspark.ml.feature import Binarizer; binarizer = Binarizer(threshold=0.5, inputCol="features", outputCol="binaryFeatures")
- Bucketizer: from pyspark.ml.feature import Bucketizer; bucketizer = Bucketizer(splits=[0, 10, 20, 30], inputCol="age", outputCol="ageBucket")
- ElementwiseProduct: from pyspark.ml.feature import ElementwiseProduct; elementwise_product = ElementwiseProduct(scalingVec=dense_vec, inputCol="features", outputCol="scaledFeatures")
- MaxAbsScaler: from pyspark.ml.feature import MaxAbsScaler; max_abs_scaler = MaxAbsScaler(inputCol="features", outputCol="scaledFeatures")
- MinMaxScaler: from pyspark.ml.feature import MinMaxScaler; min_max_scaler = MinMaxScaler(min=0.0, max=1.0, inputCol="features", outputCol="scaledFeatures")
- Normalizer: from pyspark.ml.feature import Normalizer; normalizer = Normalizer(p=2.0, inputCol="features", outputCol="normalizedFeatures")
- OneHotEncoder: from pyspark.ml.feature import OneHotEncoder; one_hot_encoder = OneHotEncoder(inputCols=["category"], outputCols=["categoryVec"])
- PolynomialExpansion: from pyspark.ml.feature import PolynomialExpansion; poly_expansion = PolynomialExpansion(degree=2, inputCol="features", outputCol="polyFeatures")
- QuantileDiscretizer: from pyspark.ml.feature import QuantileDiscretizer; quantile_discretizer = QuantileDiscretizer(numBuckets=5, inputCol="age", outputCol="ageBucket")

By: Waleed Mousa

- RobustScaler: `from pyspark.ml.feature import RobustScaler; robust_scaler = RobustScaler(withMedian=True, inputCol="features", outputCol="scaledFeatures")`
- StandardScaler: `from pyspark.ml.feature import StandardScaler; standard_scaler = StandardScaler(withMean=True, withStd=True, inputCol="features", outputCol="scaledFeatures")`
- VectorAssembler: `from pyspark.ml.feature import VectorAssembler; vector_assembler = VectorAssembler(inputCols=["col1", "col2"], outputCol="features")`
- VectorIndexer: `from pyspark.ml.feature import VectorIndexer; vector_indexer = VectorIndexer(maxCategories=5, inputCol="features", outputCol="indexedFeatures")`
- VectorSlicer: `from pyspark.ml.feature import VectorSlicer; vector_slicer = VectorSlicer(inputCol="features", outputCol="slicedFeatures", indices=[1, 3])`

3. Feature Selection

- ChiSqSelector: `from pyspark.ml.feature import ChiSqSelector; chi_sq_selector = ChiSqSelector(numTopFeatures=10, featuresCol="features", outputCol="selectedFeatures", labelCol="label")`
- UnivariateFeatureSelector: `from pyspark.ml.feature import UnivariateFeatureSelector; selector = UnivariateFeatureSelector(featuresCol="features", outputCol="selectedFeatures", labelCol="label")`
- VarianceThresholdSelector: `from pyspark.ml.feature import VarianceThresholdSelector; selector = VarianceThresholdSelector(varianceThreshold=0.5, featuresCol="features", outputCol="selectedFeatures")`

4. Model Training and Evaluation

- LogisticRegression: `from pyspark.ml.classification import LogisticRegression; lr = LogisticRegression(maxIter=10, regParam=0.01, elasticNetParam=0.8)`
- DecisionTreeClassifier: `from pyspark.ml.classification import DecisionTreeClassifier; dt = DecisionTreeClassifier(maxDepth=5, impurity="gini")`
- RandomForestClassifier: `from pyspark.ml.classification import RandomForestClassifier; rf = RandomForestClassifier(numTrees=100, maxDepth=5)`

By: Waleed Mousa

- GBTClassifier: from pyspark.ml.classification import GBTClassifier; gbt = GBTClassifier(maxIter=100, maxDepth=5)
- NaiveBayes: from pyspark.ml.classification import NaiveBayes; nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
- LinearSVC: from pyspark.ml.classification import LinearSVC; lsvc = LinearSVC(maxIter=10, regParam=0.1)
- OneVsRest: from pyspark.ml.classification import OneVsRest; ovr = OneVsRest(classifier=lr)
- LinearRegression: from pyspark.ml.regression import LinearRegression; lr = LinearRegression(maxIter=10, regParam=0.01, elasticNetParam=0.8)
- GeneralizedLinearRegression: from pyspark.ml.regression import GeneralizedLinearRegression; glr = GeneralizedLinearRegression(family="gaussian", link="identity", maxIter=10, regParam=0.1)
- DecisionTreeRegressor: from pyspark.ml.regression import DecisionTreeRegressor; dt = DecisionTreeRegressor(maxDepth=5)
- RandomForestRegressor: from pyspark.ml.regression import RandomForestRegressor; rf = RandomForestRegressor(numTrees=100, maxDepth=5)
- GBTRegressor: from pyspark.ml.regression import GBTRegressor; gbt = GBTRegressor(maxIter=100, maxDepth=5)
- AFTSurvivalRegression: from pyspark.ml.regression import AFTSurvivalRegression; aft = AFTSurvivalRegression(maxIter=100, censorCol="censor")
- IsotonicRegression: from pyspark.ml.regression import IsotonicRegression; ir = IsotonicRegression(isotonic=True, featureIndex=0, labelCol="label")
- KMeans: from pyspark.ml.clustering import KMeans; kmeans = KMeans(k=3, seed=1)
- GaussianMixture: from pyspark.ml.clustering import GaussianMixture; gmm = GaussianMixture(k=3, seed=1)
- LDA: from pyspark.ml.clustering import LDA; lda = LDA(k=3, maxIter=10)
- BisectingKMeans: from pyspark.ml.clustering import BisectingKMeans; bkm = BisectingKMeans(k=3, maxIter=10)
- FPGrowth: from pyspark.ml.fpm import FPGrowth; fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6)
- PrefixSpan: from pyspark.ml.fpm import PrefixSpan; prefixSpan = PrefixSpan(minSupport=0.1, maxPatternLength=5, maxLocalProjDBSize=32000000)
- ALSModel: from pyspark.ml.recommendation import ALS; als = ALS(rank=10, maxIter=10, regParam=0.1, userCol="userId", itemCol="movieId", ratingCol="rating")

By: Waleed Mousa

5. Model Evaluation

- BinaryClassificationEvaluator: `from pyspark.ml.evaluation import BinaryClassificationEvaluator; evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", labelCol="label", metricName="areaUnderROC")`
- MulticlassClassificationEvaluator: `from pyspark.ml.evaluation import MulticlassClassificationEvaluator; evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")`
- RegressionEvaluator: `from pyspark.ml.evaluation import RegressionEvaluator; evaluator = RegressionEvaluator(predictionCol="prediction", labelCol="label", metricName="rmse")`
- ClusteringEvaluator: `from pyspark.ml.evaluation import ClusteringEvaluator; evaluator = ClusteringEvaluator(predictionCol="prediction", featuresCol="features", metricName="silhouette")`
- RankingEvaluator: `from pyspark.ml.evaluation import RankingEvaluator; evaluator = RankingEvaluator(predictionCol="prediction", labelCol="label", metricName="meanAveragePrecision")`

6. Model Selection and Tuning

- ParamGridBuilder: `from pyspark.ml.tuning import ParamGridBuilder; param_grid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01]).addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]).build()`
- CrossValidator: `from pyspark.ml.tuning import CrossValidator; cv = CrossValidator(estimator=lr, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=3)`
- TrainValidationSplit: `from pyspark.ml.tuning import TrainValidationSplit; tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=param_grid, evaluator=evaluator, trainRatio=0.8)`

7. Model Persistence

- Save model: `model.save("path/to/model")`
- Load model: `loaded_model = LogisticRegressionModel.load("path/to/model")`
- Save pipeline: `pipeline.save("path/to/pipeline")`
- Load pipeline: `loaded_pipeline = Pipeline.load("path/to/pipeline")`

By: Waleed Mousa

8. Distributed Matrices

- Create a local matrix: `from pyspark.ml.linalg import Matrices; local_matrix = Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])`
- Create a distributed matrix: `from pyspark.ml.linalg.distributed import RowMatrix; rdd = sc.parallelize(local_matrix.toArray()); dist_matrix = RowMatrix(rdd.map(lambda x: Vectors.dense(x)))`
- Compute column summary statistics: `col_stats = dist_matrix.computeColumnSummaryStatistics()`
- Compute Gramian matrix: `gram_matrix = dist_matrix.computeGramianMatrix()`
- Compute covariance matrix: `cov_matrix = dist_matrix.computeCovariance()`
- Compute principal components: `pca = dist_matrix.computePrincipalComponents(k=3)`
- Compute singular value decomposition: `svd = dist_matrix.computeSVD(k=3, computeU=True)`

9. Pipelines

- Create a pipeline: `from pyspark.ml import Pipeline; pipeline = Pipeline(stages=[assembler, scaler, lr])`
- Fit a pipeline: `pipeline_model = pipeline.fit(train_data)`
- Transform data using a pipeline: `predictions = pipeline_model.transform(test_data)`

10. Utilities

- Correlation: `from pyspark.ml.stat import Correlation; corr_matrix = Correlation.corr(dataset, "features", "pearson")`
- ChiSquareTest: `from pyspark.ml.stat import ChiSquareTest; chi_square_test = ChiSquareTest.test(dataset, "features", "label")`
- Summarizer: `from pyspark.ml.stat import Summarizer; summary = Summarizer.metrics("mean", "variance").summary(dataset)`
- MulticlassMetrics: `from pyspark.mllib.evaluation import MulticlassMetrics; metrics = MulticlassMetrics(predictions.select("prediction", "label").rdd)`
- BinaryClassificationMetrics: `from pyspark.mllib.evaluation import BinaryClassificationMetrics; metrics = BinaryClassificationMetrics(predictions.select("rawPrediction", "label").rdd)`

By: Waleed Mousa

- RegressionMetrics: `from pyspark.mllib.evaluation import RegressionMetrics; metrics = RegressionMetrics(predictions.select("prediction", "label").rdd)`
- RankingMetrics: `from pyspark.mllib.evaluation import RankingMetrics; metrics = RankingMetrics(predictions.select("prediction", "label").rdd)`

11. Optimization

- Stochastic Gradient Descent (SGD): `from pyspark.ml.optimization import GradientDescent; sgd = GradientDescent(stepSize=0.1, numIterations=10)`
- Limited-memory BFGS (L-BFGS): `from pyspark.ml.optimization import LBFGS; lbfgs = LBFGS(maxIter=10, numCorrections=5)`
- Accelerated Gradient Descent (AGD): `from pyspark.ml.optimization import AcceleratedGradientDescent; agd = AcceleratedGradientDescent(stepSize=0.1, numIterations=10)`

12. Dimensionality Reduction

- PCA: `from pyspark.ml.feature import PCA; pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")`
- SVD: `from pyspark.ml.feature import VectorSlicer; slicer = VectorSlicer(inputCol="features", outputCol="projectedFeatures", indices=list(range(3)))`
- ICA: `from pyspark.ml.feature import ICA; ica = ICA(k=3, inputCol="features", outputCol="icaFeatures")`

13. Feature Hashing

- FeatureHasher: `from pyspark.ml.feature import FeatureHasher; hasher = FeatureHasher(inputCols=["col1", "col2"], outputCol="hashedFeatures", numFeatures=1000)`
- HashingTF: `from pyspark.ml.feature import HashingTF; hashingTF = HashingTF(inputCol="words", outputCol="features", numFeatures=1000)`

14. Text Analytics

- Tokenizer: `from pyspark.ml.feature import Tokenizer; tokenizer = Tokenizer(inputCol="text", outputCol="words")`
- RegexTokenizer: `from pyspark.ml.feature import RegexTokenizer; regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="\\W")`

By: Waleed Mousa

15. Recommender Systems

- ALS: `from pyspark.ml.recommendation import ALS; als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating")`
- User-based Collaborative Filtering: `from pyspark.ml.recommendation import ALS; userRecs = model.recommendForAllUsers(10)`
- Item-based Collaborative Filtering: `from pyspark.ml.recommendation import ALS; itemRecs = model.recommendForAllItems(10)`
- Popularity-based Recommendations: `from pyspark.sql.functions import count; popularity = ratings.groupBy("movieId").agg(count("userId").alias("count")).orderBy(desc("count"))`

16. Frequent Pattern Mining

- FP-Growth: `from pyspark.ml.fpm import FPGrowth; fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6)`
- PrefixSpan: `from pyspark.ml.fpm import PrefixSpan; prefixSpan = PrefixSpan(minSupport=0.1, maxPatternLength=5, maxLocalProjDBSize=32000000)`
- Association Rules: `from pyspark.ml.fpm import FPGrowth; fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6); model = fpGrowth.fit(data); associationRules = model.associationRules`

17. Model Interpretability

- Feature Importance: `model.featureImportances`
- Decision Tree Visualization: `from pyspark.ml.classification import DecisionTreeClassificationModel; model.toDebugString`
- Linear Model Coefficients: `model.coefficients`
- Linear Model Intercept: `model.intercept`

18. Hyperparameter Tuning

- ParamGridBuilder: `from pyspark.ml.tuning import ParamGridBuilder; paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01]).addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]).build()`
- TrainValidationSplit: `from pyspark.ml.tuning import TrainValidationSplit; tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator, trainRatio=0.8)`

By: Waleed Mousa

- CrossValidator: `from pyspark.ml.tuning import CrossValidator; cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=3)`

19. Model Evaluation Metrics

- Accuracy: `from pyspark.ml.evaluation import MulticlassClassificationEvaluator; accuracy = evaluator.evaluate(predictions, {evaluator.metricName: "accuracy"})`
- Precision: `from pyspark.ml.evaluation import MulticlassClassificationEvaluator; precision = evaluator.evaluate(predictions, {evaluator.metricName: "weightedPrecision"})`
- Recall: `from pyspark.ml.evaluation import MulticlassClassificationEvaluator; recall = evaluator.evaluate(predictions, {evaluator.metricName: "weightedRecall"})`
- F1-Score: `from pyspark.ml.evaluation import MulticlassClassificationEvaluator; f1 = evaluator.evaluate(predictions, {evaluator.metricName: "f1"})`
- Area Under ROC (AUC): `from pyspark.ml.evaluation import BinaryClassificationEvaluator; auc = evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})`
- Root Mean Squared Error (RMSE): `from pyspark.ml.evaluation import RegressionEvaluator; rmse = evaluator.evaluate(predictions, {evaluator.metricName: "rmse"})`
- Mean Absolute Error (MAE): `from pyspark.ml.evaluation import RegressionEvaluator; mae = evaluator.evaluate(predictions, {evaluator.metricName: "mae"})`
- R-squared (R2): `from pyspark.ml.evaluation import RegressionEvaluator; r2 = evaluator.evaluate(predictions, {evaluator.metricName: "r2"})`
- Silhouette Score: `from pyspark.ml.evaluation import ClusteringEvaluator; silhouette = evaluator.evaluate(predictions)`
- Mean Average Precision (MAP): `from pyspark.ml.evaluation import RankingEvaluator; map = evaluator.evaluate(predictions, {evaluator.metricName: "meanAveragePrecision"})`