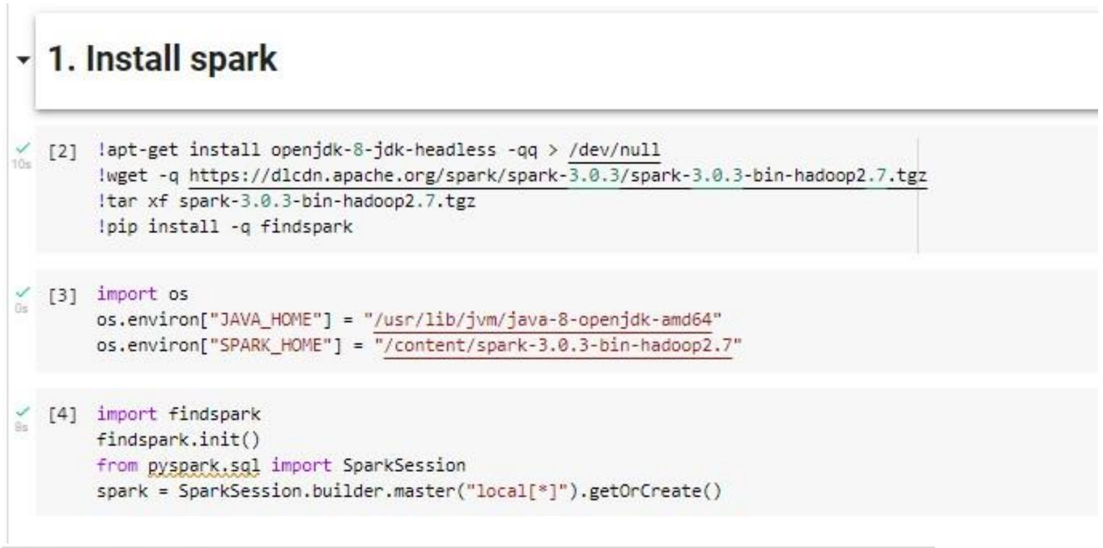
Installing Spark and Loading dataset:

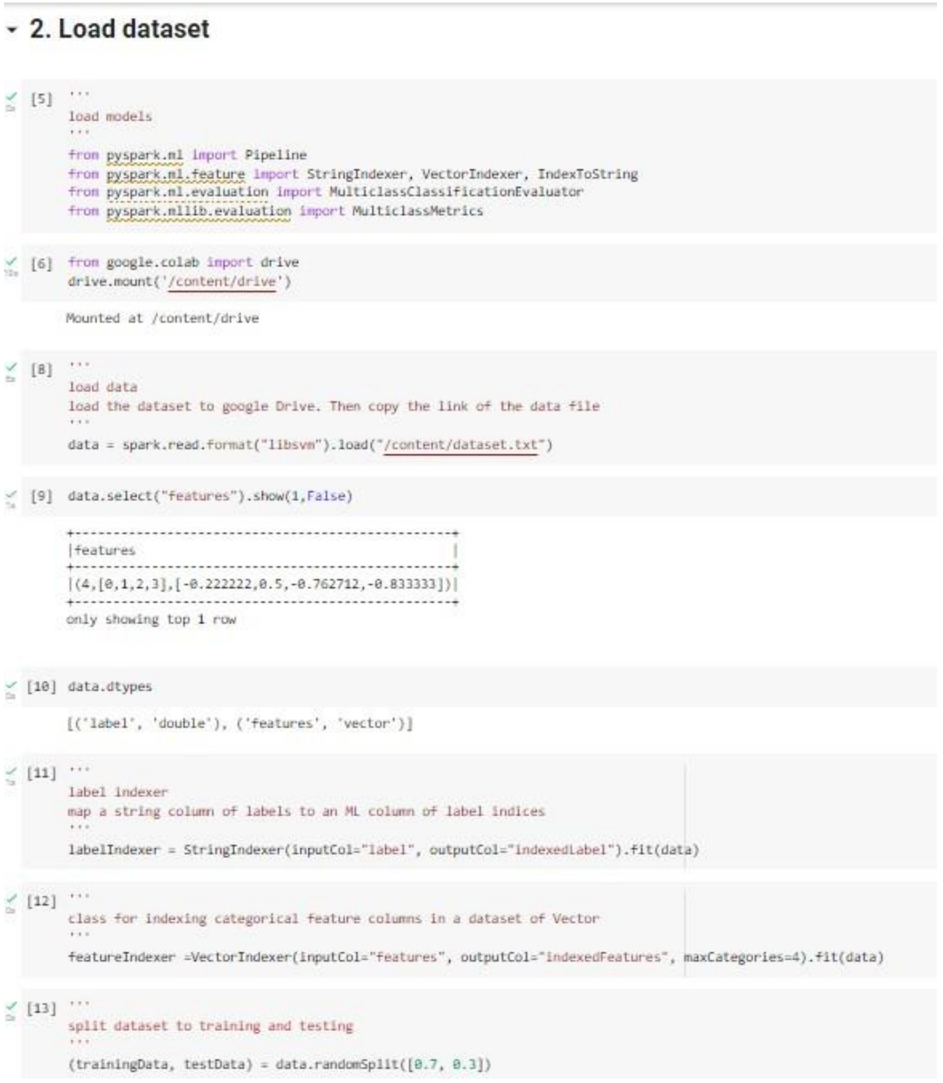


1. Install spark

```
[2] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
    !wget -q https://dlcdn.apache.org/spark/spark-3.0.3/spark-3.0.3-bin-hadoop2.7.tgz
    !tar xf spark-3.0.3-bin-hadoop2.7.tgz
    !pip install -q findspark
```

```
[3] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop2.7"
```

```
[4] import findspark
    findspark.init()
    from pyspark.sql import SparkSession
    spark = SparkSession.builder.master("local[*]").getOrCreate()
```



2. Load dataset

```
[5] '''
    load models
    '''
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, VectorIndexer, IndexToString
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    from pyspark.mllib.evaluation import MulticlassMetrics
```

```
[6] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

```
[8] '''
    load data
    load the dataset to google Drive. Then copy the link of the data file
    '''
    data = spark.read.format("libsvm").load("/content/dataset.txt")
```

```
[9] data.select("features").show(1,False)

    +----------------------------------------------+
    |features                                      |
    +----------------------------------------------+
    |(4,[0,1,2,3],[-0.222222,0.5,-0.762712,-0.833333])|
    +----------------------------------------------+
    only showing top 1 row
```

```
[10] data.dtypes

    [('label', 'double'), ('features', 'vector')]
```

```
[11] '''
    label indexer
    map a string column of labels to an ML column of label indices
    '''
    labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
```

```
[12] '''
    class for indexing categorical feature columns in a dataset of Vector
    '''
    featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
```

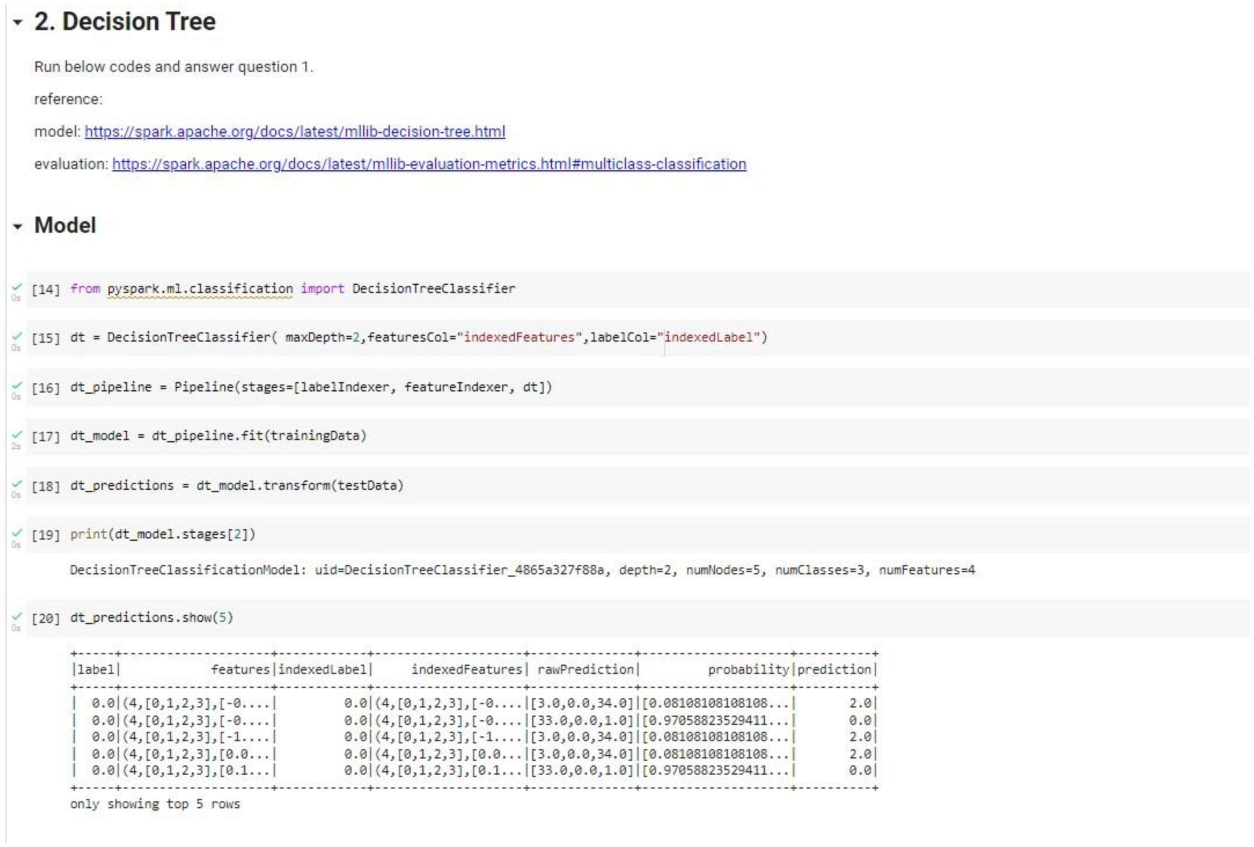
```
[13] '''
    split dataset to training and testing
    '''
    (trainingData, testData) = data.randomSplit([0.7, 0.3])
```

- Firstly we install spark version 3.0.3 and java jdk 8.
- Then we set the environment variables for spark and java.
- Then after we load the dataset.
- Here we load machine learning models like pipeline and features like StringIndexer, VectorIndexer, IndexToString and some more.
- Then we mount the contents of Google drive.
- Then we load the dataset from the Google drive.
- Here we use select and show conmand to view features of the dataset.
- Then we check the datatypes of the data using data.dtypes.
- Then we use label indexer for mapping of a string column of labels to machine learning column of indexed labels.
- Then we use feature indexer for mapping of categorical feature columns in the dataset of vector to machine learning column of indexed features.
- Then we split them dataset for the purpose of training and testing using randomSplit function.

Decision tree model:

- Decision trees are used for classification and also for regression problems.
- These are constructed based on an approach that finds ways to split dataset based on different parameters. The goal is to create a decision model that predicts the value of the target by learning rules that are grasped from the features.
- In decision tree, the deeper the tree, the complex the rules and the fitter the model.
- The decision tree makes decisions based on the set of features present in the data and the sequence of features to be checked depends on criteria like gini index or information gain.

Running the model:



2. Decision Tree

Run below codes and answer question 1.

reference:

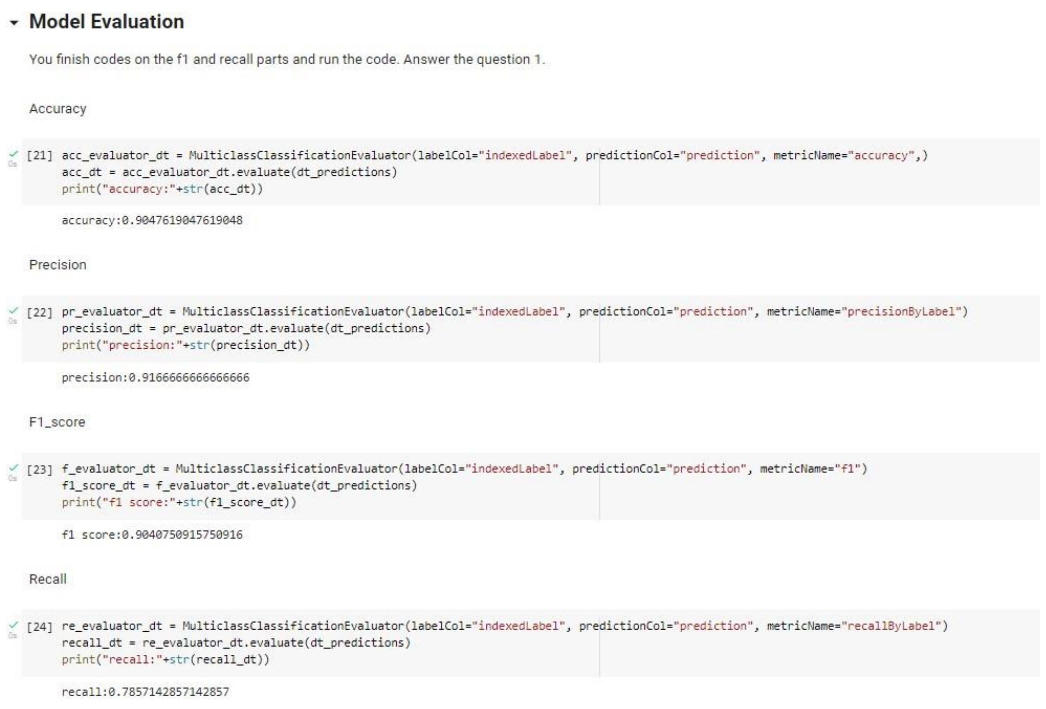
model: https://spark.apache.org/docs/latest/mllib-decision-tree.html

evaluation: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#multiclass-classification

Model

```
[14] from pyspark.ml.classification import DecisionTreeClassifier

[15] dt = DecisionTreeClassifier( maxDepth=2,featuresCol="indexedFeatures",labelCol="indexedLabel")

[16] dt_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

[17] dt_model = dt_pipeline.fit(trainingData)

[18] dt_predictions = dt_model.transform(testData)

[19] print(dt_model.stages[2])

     DecisionTreeClassificationModel: uid=DecisionTreeClassifier_4865a327f88a, depth=2, numNodes=5, numClasses=3, numFeatures=4

[20] dt_predictions.show(5)

     +-----+--------------------+------------+--------------------+--------------+--------------------+----------+
     |label|            features|indexedLabel|     indexedFeatures| rawPrediction|         probability|prediction|
     +-----+--------------------+------------+--------------------+--------------+--------------------+----------+
     |  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[3.0,0.0,34.0]|[0.08108108108108...|       2.0|
     |  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[33.0,0.0,1.0]|[0.97058823529411...|       0.0|
     |  0.0|(4,[0,1,2,3],[-1....|         0.0|(4,[0,1,2,3],[-1....|[3.0,0.0,34.0]|[0.08108108108108...|       2.0|
     |  0.0|(4,[0,1,2,3],[0.0...|         0.0|(4,[0,1,2,3],[0.0...|[3.0,0.0,34.0]|[0.08108108108108...|       2.0|
     |  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[33.0,0.0,1.0]|[0.97058823529411...|       0.0|
     +-----+--------------------+------------+--------------------+--------------+--------------------+----------+
     only showing top 5 rows
```

- Here we import the decision tree classifier and set the depth as 2 and then we fit the model with training data.
- Then we set predictions to transform the testing data and print the model stages.
- Then we display the predictions using show().



Model Evaluation

You finish codes on the f1 and recall parts and run the code. Answer the question 1.

Accuracy

```
[21] acc_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_dt = acc_evaluator_dt.evaluate(dt_predictions)
     print("accuracy:"+str(acc_dt))

     accuracy:0.9047619047619048
```

Precision

```
[22] pr_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_dt = pr_evaluator_dt.evaluate(dt_predictions)
     print("precision:"+str(precision_dt))

     precision:0.9166666666666666
```

F1_score

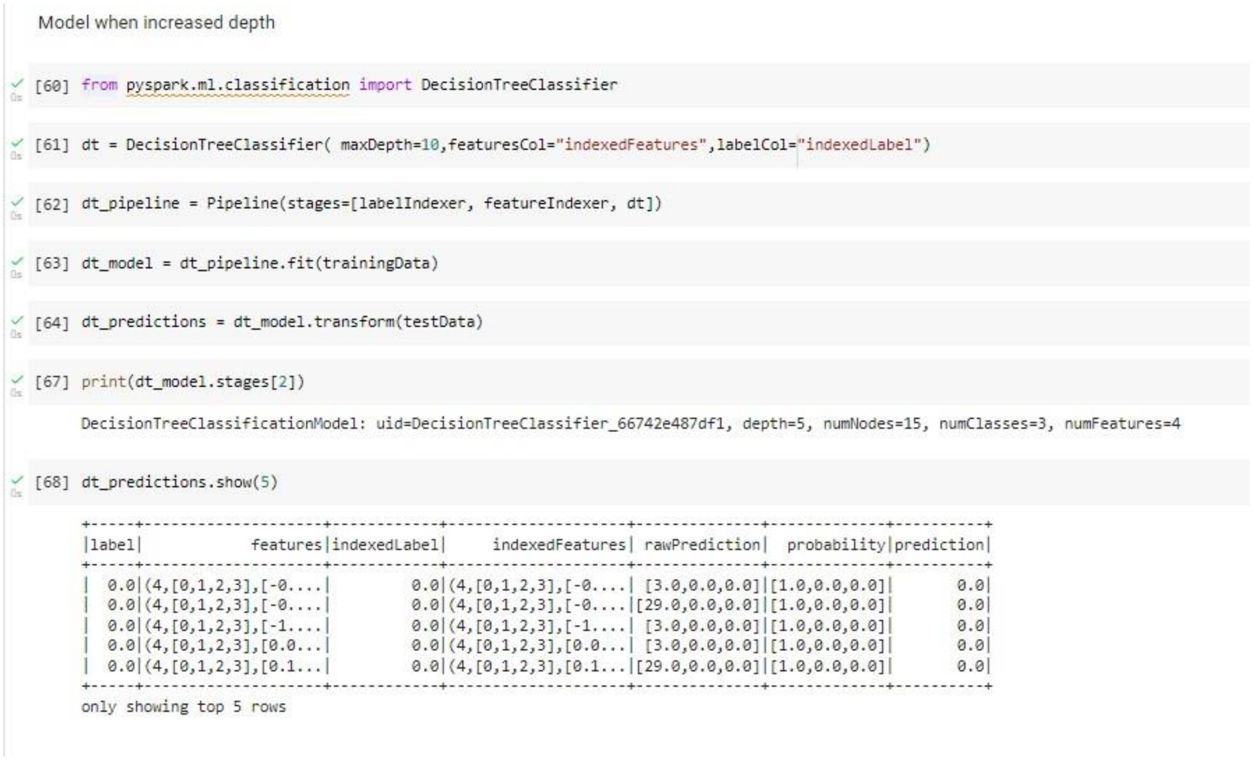
```
[23] f_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_dt = f_evaluator_dt.evaluate(dt_predictions)
     print("f1 score:"+str(f1_score_dt))

     f1 score:0.9040750915750916
```

Recall

```
[24] re_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_dt = re_evaluator_dt.evaluate(dt_predictions)
     print("recall:"+str(recall_dt))

     recall:0.7857142857142857
```

- Then we check the values of accuracy, precision, f1 score and recall.
- We note down the values of all the results.
- Now we change the max depth and check the values again.

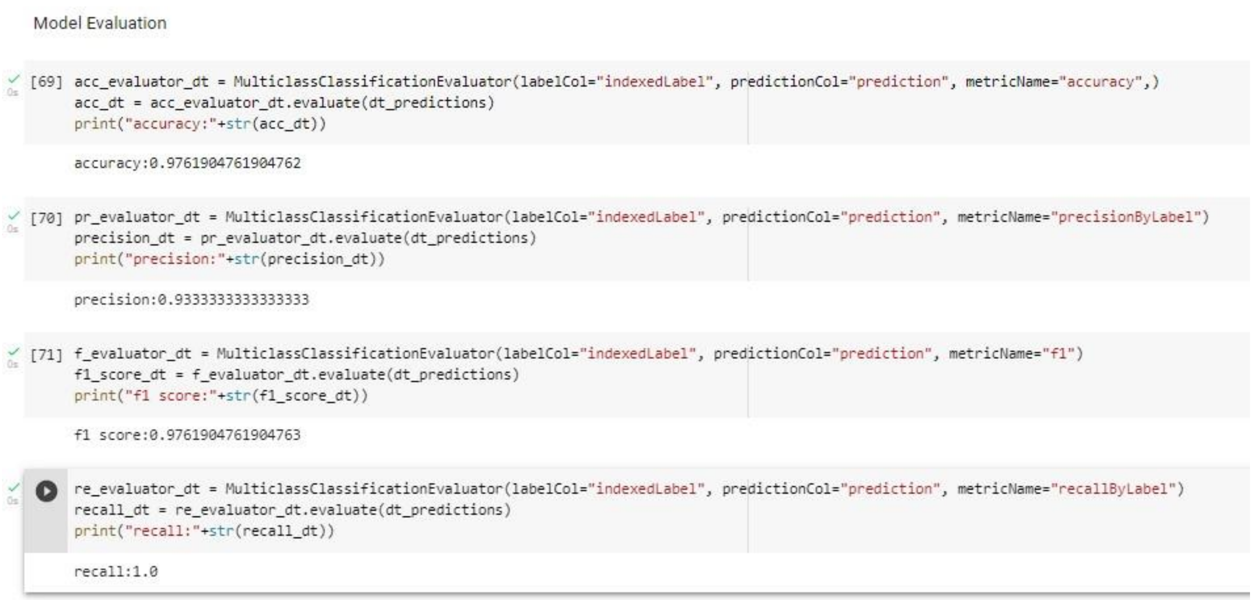
Model after increasing depth



Model when increased depth

```
[60] from pyspark.ml.classification import DecisionTreeClassifier

[61] dt = DecisionTreeClassifier( maxDepth=10,featuresCol="indexedFeatures",labelCol="indexedLabel")

[62] dt_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

[63] dt_model = dt_pipeline.fit(trainingData)

[64] dt_predictions = dt_model.transform(testData)

[67] print(dt_model.stages[2])

     DecisionTreeClassificationModel: uid=DecisionTreeClassifier_66742e487df1, depth=5, numNodes=15, numClasses=3, numFeatures=4

[68] dt_predictions.show(5)

     +-----+--------------------+------------+--------------------+--------------+-------------+----------+
     |label|            features|indexedLabel|     indexedFeatures| rawPrediction|  probability|prediction|
     +-----+--------------------+------------+--------------------+--------------+-------------+----------+
     |  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....| [3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
     |  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[29.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
     |  0.0|(4,[0,1,2,3],[-1....|         0.0|(4,[0,1,2,3],[-1....| [3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
     |  0.0|(4,[0,1,2,3],[0.0...|         0.0|(4,[0,1,2,3],[0.0...| [3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
     |  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[29.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
     +-----+--------------------+------------+--------------------+--------------+-------------+----------+
     only showing top 5 rows
```

- Here we increase the max depth to 10 and then run the model again.



Model Evaluation

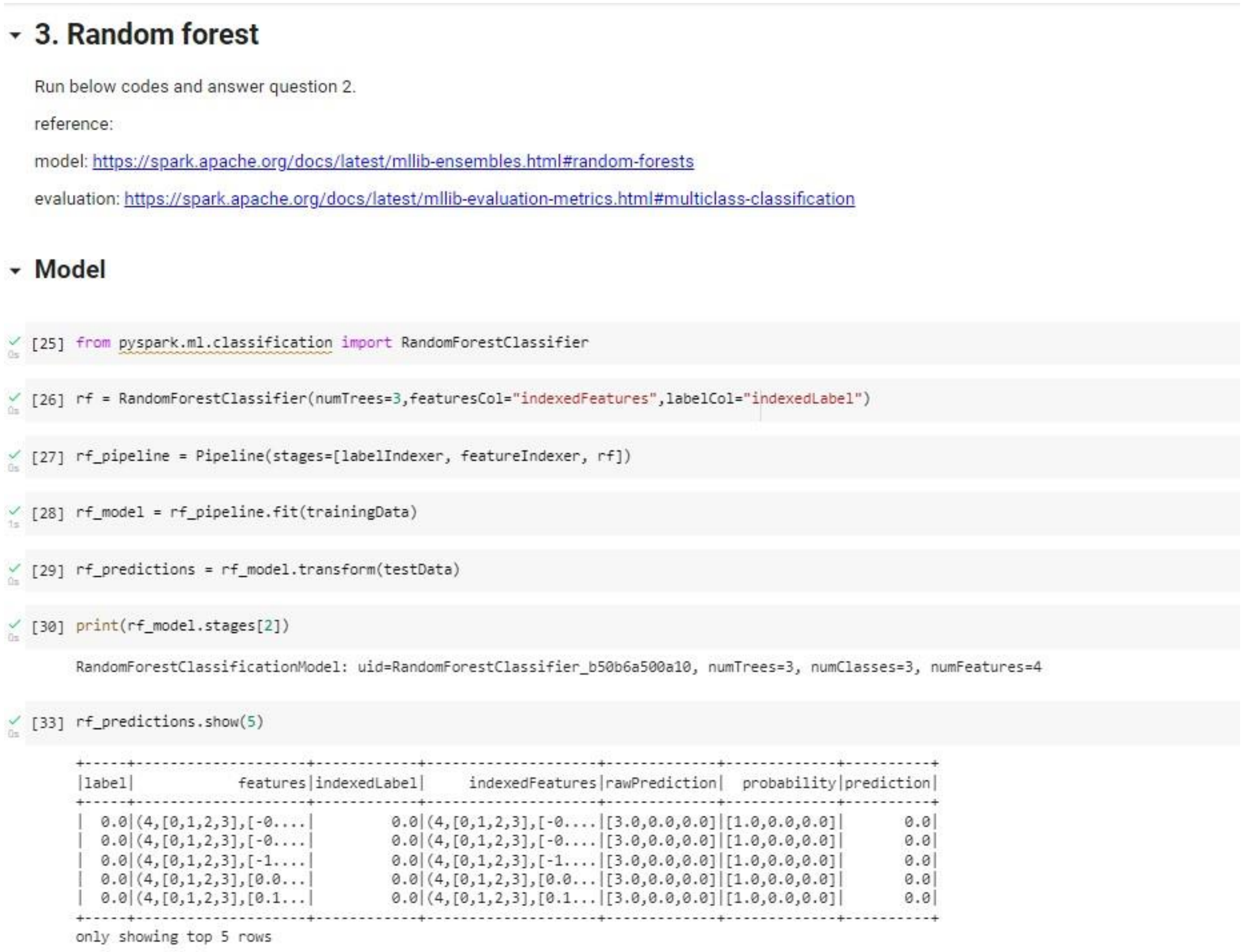
```
[69] acc_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_dt = acc_evaluator_dt.evaluate(dt_predictions)
     print("accuracy:"+str(acc_dt))

     accuracy:0.9761904761904762

[70] pr_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_dt = pr_evaluator_dt.evaluate(dt_predictions)
     print("precision:"+str(precision_dt))

     precision:0.9333333333333333

[71] f_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_dt = f_evaluator_dt.evaluate(dt_predictions)
     print("f1 score:"+str(f1_score_dt))

     f1 score:0.9761904761904763

     re_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_dt = re_evaluator_dt.evaluate(dt_predictions)
     print("recall:"+str(recall_dt))

     recall:1.0
```

- We can see that the values of accuracy, precision, f1 score were increased and thus we can assure that increase in the max depth can increase the performance of the decision tree model.

Random Forest Model:

- Random Forest algorithm leverages the performance of multiple decision trees to make decisions. It calculates output by working on random set of features and then combines the result of individual decision trees to produce the output.
- If we increase the number of trees in random forest, the model's accuracy improves because of decrease in variance in predictions.
- Random forest has high training time than decision tree because as we increase the number of trees the time taken for training also increases.

Running the model:



3. Random forest

Run below codes and answer question 2.

reference:

model: https://spark.apache.org/docs/latest/mllib-ensembles.html#random-forests

evaluation: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#multiclass-classification

Model

```
[25] from pyspark.ml.classification import RandomForestClassifier
```

```
[26] rf = RandomForestClassifier(numTrees=3,featuresCol="indexedFeatures",labelCol="indexedLabel")
```

```
[27] rf_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf])
```

```
[28] rf_model = rf_pipeline.fit(trainingData)
```

```
[29] rf_predictions = rf_model.transform(testData)
```

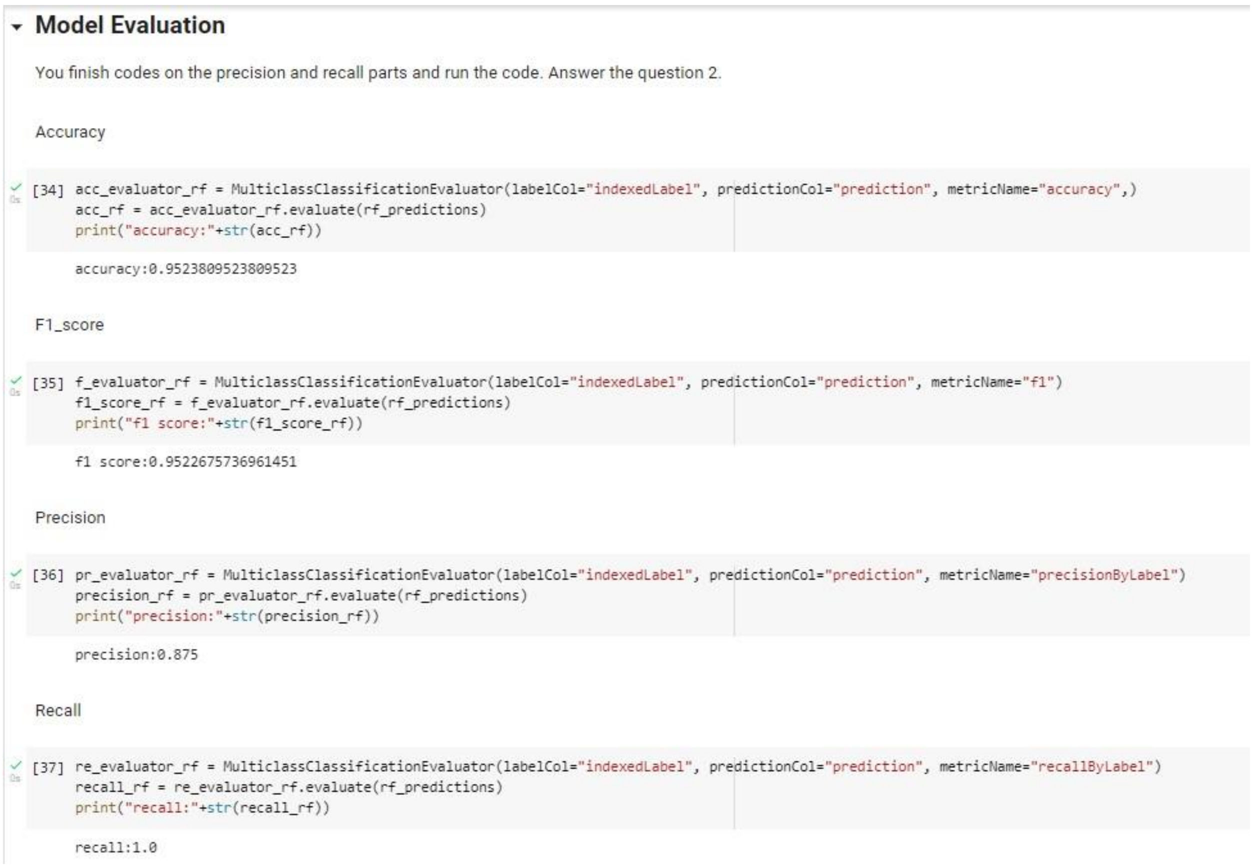
```
[30] print(rf_model.stages[2])
```

```
RandomForestClassificationModel: uid=RandomForestClassifier_b50b6a500a10, numTrees=3, numClasses=3, numFeatures=4
```

```
[33] rf_predictions.show(5)
```

```
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
|label|            features|indexedLabel|     indexedFeatures|rawPrediction|  probability|prediction|
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[-1....|         0.0|(4,[0,1,2,3],[-1....|[3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.0...|         0.0|(4,[0,1,2,3],[0.0...|[3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[3.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
only showing top 5 rows
```

- Here we import the random forest classifier and set the number of trees as 3and then we fit the model with training data.
- Then we set predictions to transform the testing data and print the model stages.
- Then we display the predictions using show().



Model Evaluation

You finish codes on the precision and recall parts and run the code. Answer the question 2.

Accuracy

```
[34] acc_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_rf = acc_evaluator_rf.evaluate(rf_predictions)
     print("accuracy:"+str(acc_rf))
```

```
accuracy:0.9523809523809523
```

F1_score

```
[35] f_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_rf = f_evaluator_rf.evaluate(rf_predictions)
     print("f1 score:"+str(f1_score_rf))
```

```
f1 score:0.9522675736961451
```

Precision

```
[36] pr_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_rf = pr_evaluator_rf.evaluate(rf_predictions)
     print("precision:"+str(precision_rf))
```

```
precision:0.875
```

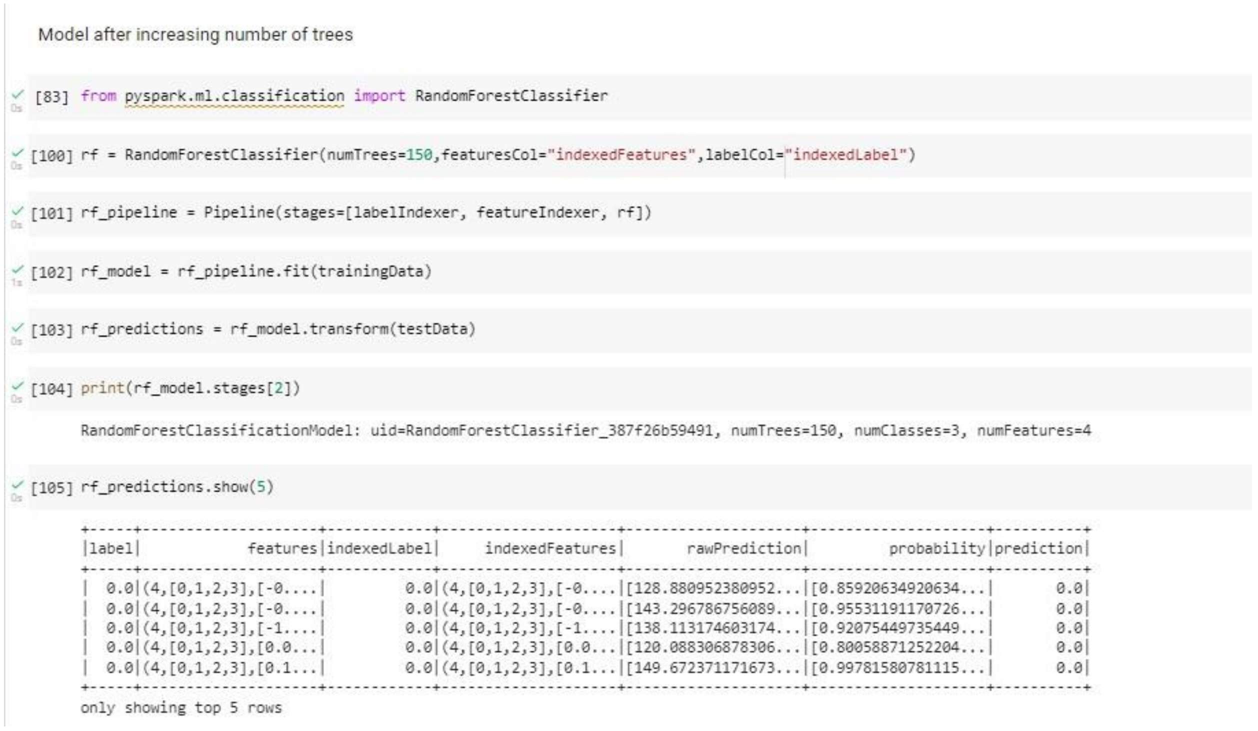
Recall

```
[37] re_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_rf = re_evaluator_rf.evaluate(rf_predictions)
     print("recall:"+str(recall_rf))
```

```
recall:1.0
```

- Then we check the values of accuracy, precision, f1 score and recall.
- We note down the values of all the results.
- Now we change the number of trees and check the values again.

Model after increasing depth



Model after increasing number of trees

```
[83] from pyspark.ml.classification import RandomForestClassifier
```

```
[100] rf = RandomForestClassifier(numTrees=150,featuresCol="indexedFeatures",labelCol="indexedLabel")
```

```
[101] rf_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf])
```

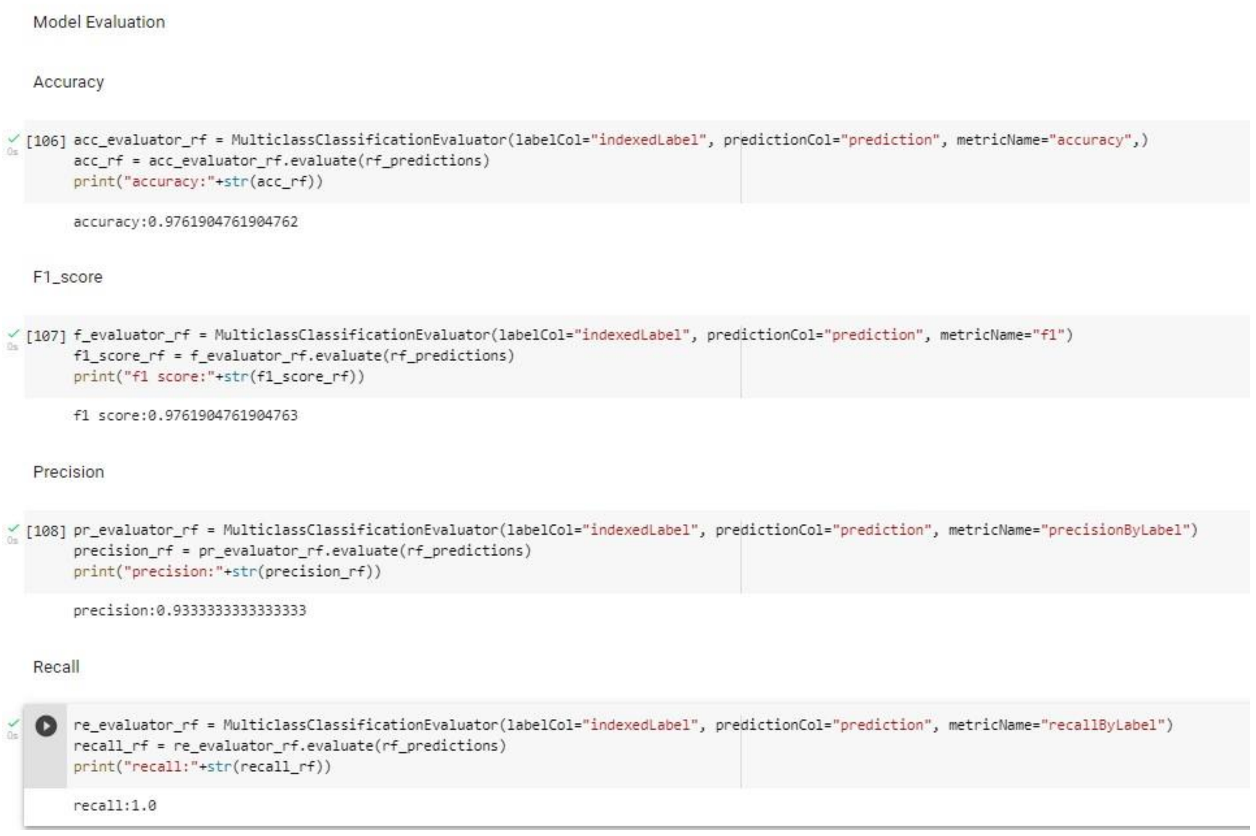
```
[102] rf_model = rf_pipeline.fit(trainingData)
```

```
[103] rf_predictions = rf_model.transform(testData)
```

```
[104] print(rf_model.stages[2])

RandomForestClassificationModel: uid=RandomForestClassifier_387f26b59491, numTrees=150, numClasses=3, numFeatures=4
```

```
[105] rf_predictions.show(5)

+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
|label|            features|indexedLabel|     indexedFeatures|       rawPrediction|         probability|prediction|
+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[128.880952380952...|[0.85920634920634...|       0.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[143.296786756089...|[0.95531191170726...|       0.0|
|  0.0|(4,[0,1,2,3],[-1....|         0.0|(4,[0,1,2,3],[-1....|[138.113174603174...|[0.92075449735449...|       0.0|
|  0.0|(4,[0,1,2,3],[0.0...|         0.0|(4,[0,1,2,3],[0.0...|[120.088306878306...|[0.80058871252204...|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[149.672371171673...|[0.99781580781115...|       0.0|
+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

- Here we increase number of trees to 150 and then run the model again.



Model Evaluation

Accuracy

```
[106] acc_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
      acc_rf = acc_evaluator_rf.evaluate(rf_predictions)
      print("accuracy:"+str(acc_rf))

accuracy:0.9761904761904762
```

F1_score

```
[107] f_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
      f1_score_rf = f_evaluator_rf.evaluate(rf_predictions)
      print("f1 score:"+str(f1_score_rf))

f1 score:0.9761904761904763
```

Precision

```
[108] pr_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
      precision_rf = pr_evaluator_rf.evaluate(rf_predictions)
      print("precision:"+str(precision_rf))

precision:0.9333333333333333
```

Recall

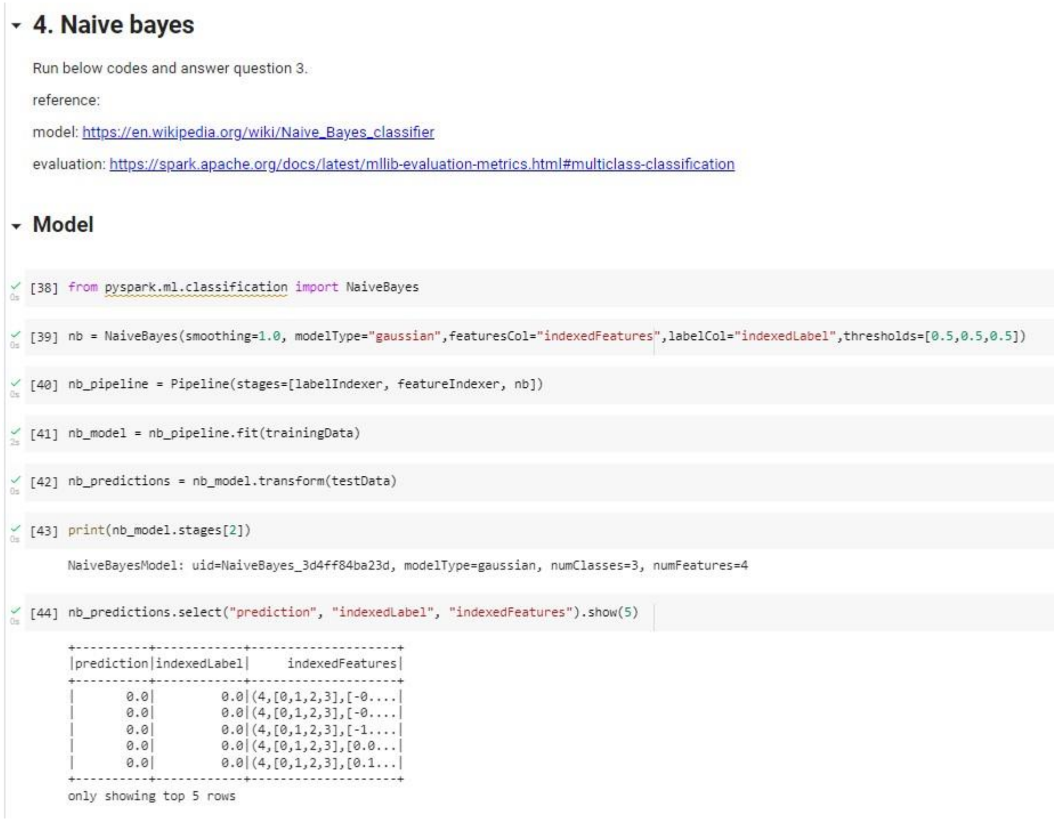
```
re_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
recall_rf = re_evaluator_rf.evaluate(rf_predictions)
print("recall:"+str(recall_rf))

recall:1.0
```

- We can see that the values of accuracy, precision, f1 score were increased and thus we can assure that increase in the number of trees can increase the performance of the random forest model.

Naïve Bayes Model:

- Naive Bayes algorithm works on bayes theorem and it is mostly used in classification problems.
- This model assumes that the features that are present in the model are independent of each other so that changing value of one feature does not influence other features.
- The class with higher posterior probability is the output of prediction.
- It performs well in the case of categorical variables when compared to numerical variables.

Running the model:



4. Naive bayes

Run below codes and answer question 3.

reference:

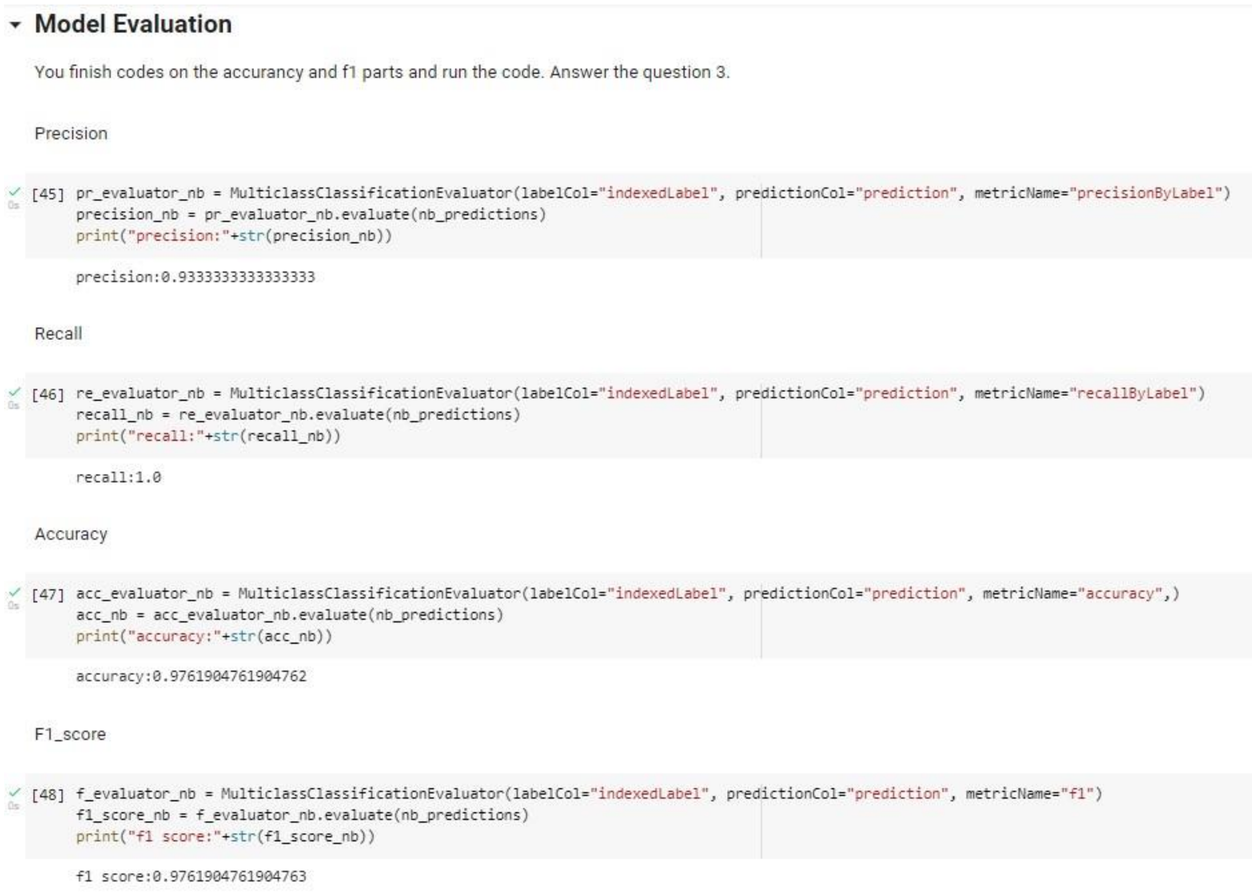
model: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

evaluation: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#multiclass-classification

Model

```
[38] from pyspark.ml.classification import NaiveBayes

[39] nb = NaiveBayes(smoothing=1.0, modelType="gaussian",featuresCol="indexedFeatures",labelCol="indexedLabel",thresholds=[0.5,0.5,0.5])

[40] nb_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, nb])

[41] nb_model = nb_pipeline.fit(trainingData)

[42] nb_predictions = nb_model.transform(testData)

[43] print(nb_model.stages[2])

     NaiveBayesModel: uid=NaiveBayes_3d4ff84ba23d, modelType=gaussian, numClasses=3, numFeatures=4

[44] nb_predictions.select("prediction", "indexedLabel", "indexedFeatures").show(5)

     +----------+------------+--------------------+
     |prediction|indexedLabel|     indexedFeatures|
     +----------+------------+--------------------+
     |       0.0|         0.0|(4,[0,1,2,3],[-0....|
     |       0.0|         0.0|(4,[0,1,2,3],[-0....|
     |       0.0|         0.0|(4,[0,1,2,3],[-1....|
     |       0.0|         0.0|(4,[0,1,2,3],[0.0...|
     |       0.0|         0.0|(4,[0,1,2,3],[0.1...|
     +----------+------------+--------------------+
     only showing top 5 rows
```

- Here we import the Naïve Bayes classifier and set the thresholds as 0.5 and then we fit the model with training data.
- Then we set predictions to transform the testing data and print the model stages.
- Then we display the predictions using show().



Model Evaluation

You finish codes on the accurancy and f1 parts and run the code. Answer the question 3.

Precision

```
[45] pr_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_nb = pr_evaluator_nb.evaluate(nb_predictions)
     print("precision:"+str(precision_nb))
```

```
precision:0.9333333333333333
```

Recall

```
[46] re_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_nb = re_evaluator_nb.evaluate(nb_predictions)
     print("recall:"+str(recall_nb))
```

```
recall:1.0
```

Accuracy

```
[47] acc_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_nb = acc_evaluator_nb.evaluate(nb_predictions)
     print("accuracy:"+str(acc_nb))
```

```
accuracy:0.9761904761904762
```

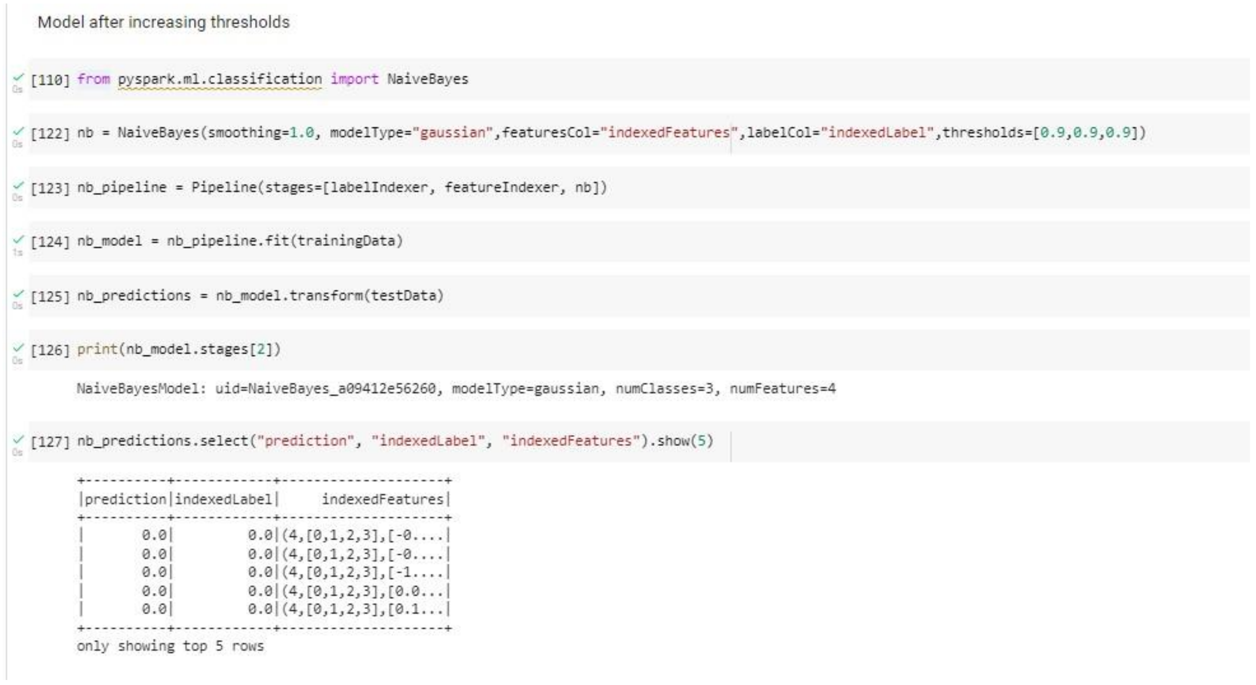
F1_score

```
[48] f_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_nb = f_evaluator_nb.evaluate(nb_predictions)
     print("f1 score:"+str(f1_score_nb))
```

```
f1 score:0.9761904761904763
```

- Then we check the values of accuracy, precision, f1 score and recall.
- We note down the values of all the results.
- Now we change the threshold values and check the values again.

Model after increasing thresholds:



Model after increasing thresholds

```
[110] from pyspark.ml.classification import NaiveBayes
```

```
[122] nb = NaiveBayes(smoothing=1.0, modelType="gaussian",featuresCol="indexedFeatures",labelCol="indexedLabel",thresholds=[0.9,0.9,0.9])
```

```
[123] nb_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, nb])
```

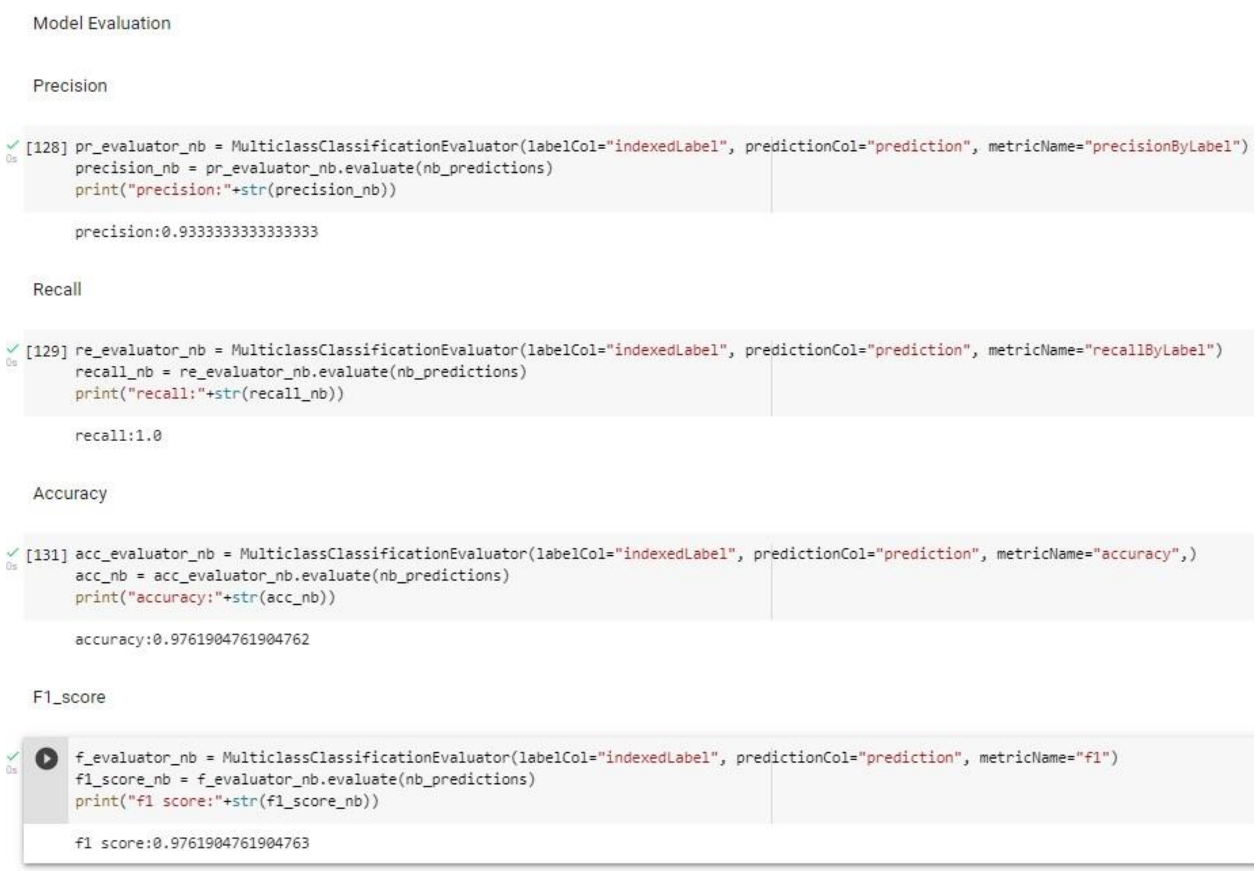
```
[124] nb_model = nb_pipeline.fit(trainingData)
```

```
[125] nb_predictions = nb_model.transform(testData)
```

```
[126] print(nb_model.stages[2])

NaiveBayesModel: uid=NaiveBayes_a09412e56260, modelType=gaussian, numClasses=3, numFeatures=4
```

```
[127] nb_predictions.select("prediction", "indexedLabel", "indexedFeatures").show(5)

+----------+------------+--------------------+
|prediction|indexedLabel|     indexedFeatures|
+----------+------------+--------------------+
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-1....|
|       0.0|         0.0|(4,[0,1,2,3],[0.0...|
|       0.0|         0.0|(4,[0,1,2,3],[0.1...|
+----------+------------+--------------------+
only showing top 5 rows
```

- Here we increase thresholds to 0.9 and then run the model again.



Model Evaluation

Precision

```
[128] pr_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
      precision_nb = pr_evaluator_nb.evaluate(nb_predictions)
      print("precision:"+str(precision_nb))

      precision:0.9333333333333333
```

Recall

```
[129] re_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
      recall_nb = re_evaluator_nb.evaluate(nb_predictions)
      print("recall:"+str(recall_nb))

      recall:1.0
```

Accuracy

```
[131] acc_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
      acc_nb = acc_evaluator_nb.evaluate(nb_predictions)
      print("accuracy:"+str(acc_nb))

      accuracy:0.9761904761904762
```

F1_score

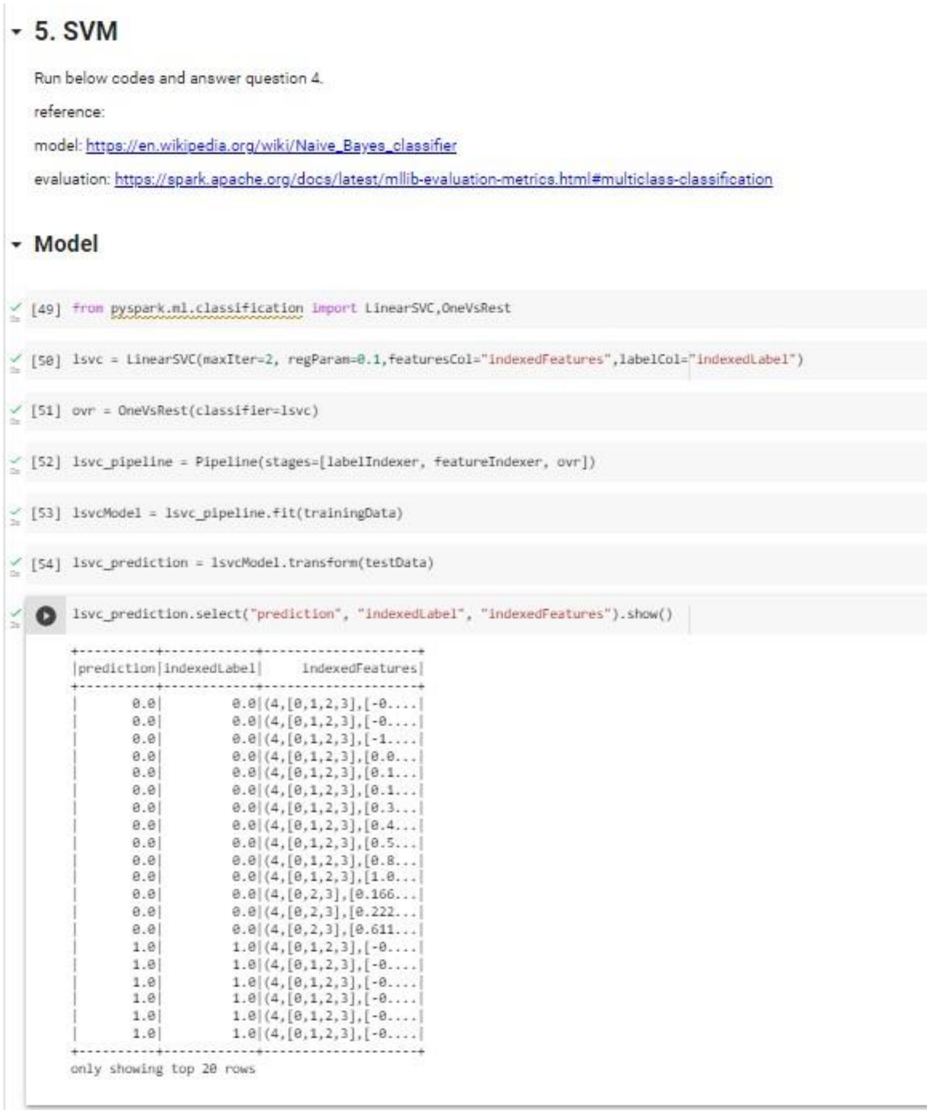
```
      f_evaluator_nb = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
      f1_score_nb = f_evaluator_nb.evaluate(nb_predictions)
      print("f1 score:"+str(f1_score_nb))

      f1 score:0.9761904761904763
```

- We can see that the values of accuracy, precision, f1 score were increased and thus we can assure that increase in the thresholds value can increase the performance of the naïve bayes model.

SVM model :

- A support vector machine is a supervised machine learning model that utilizes classification algorithms for classification problems.
- This model is best for text classification problems.
- In SPM algorithm we plot each training data object as a point in feature dimensional space With value of each feature being the value of appropriate coordinate.
- then we perform classification by identifying the hyperplane that separates the two classes

Running the model



5. SVM

Run below codes and answer question 4.

reference:

model: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

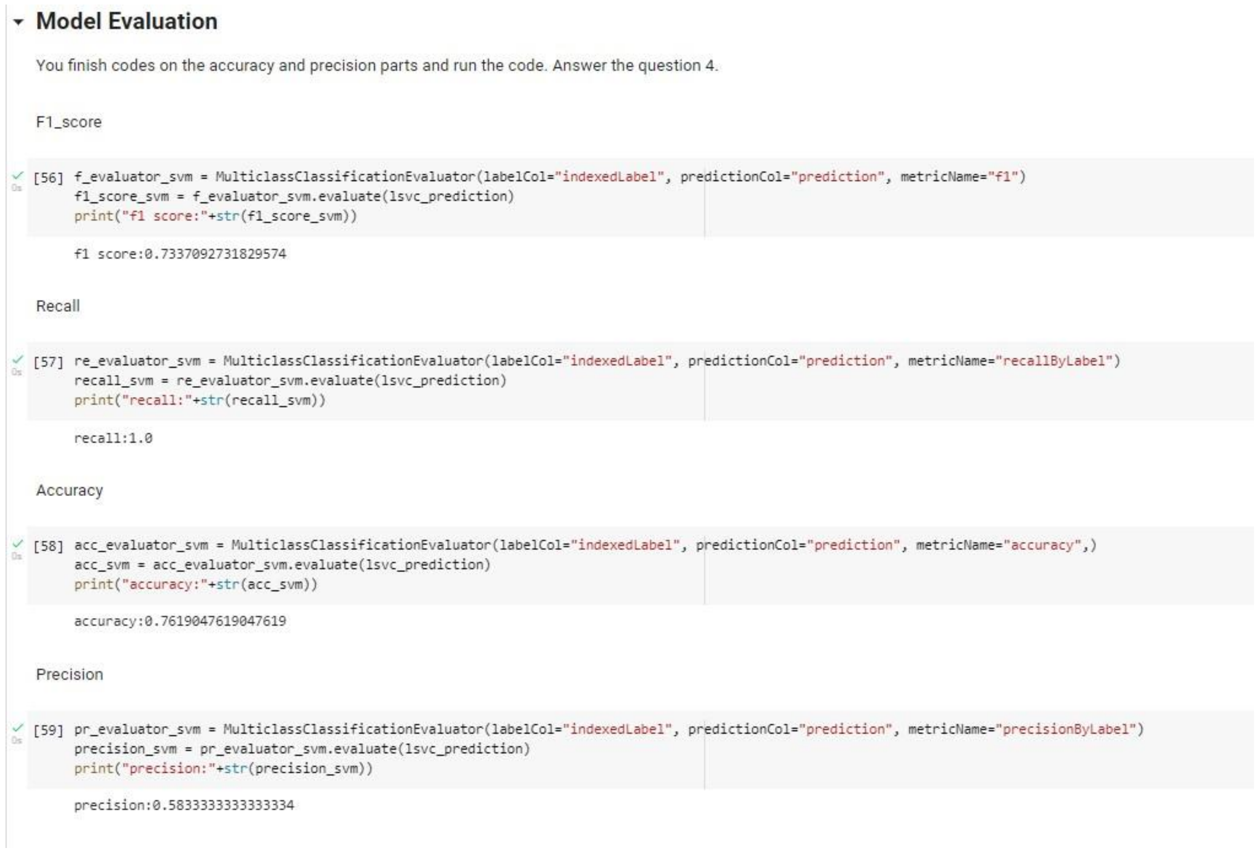
evaluation: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#multiclass-classification

Model

```
[49] from pyspark.ml.classification import LinearSVC,OneVsRest
[50] lsvc = LinearSVC(maxIter=2, regParam=0.1,featuresCol="indexedFeatures",labelCol="indexedLabel")
[51] ovr = OneVsRest(classifier=lsvc)
[52] lsvc_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, ovr])
[53] lsvcModel = lsvc_pipeline.fit(trainingData)
[54] lsvc_prediction = lsvcModel.transform(testData)
lsvc_prediction.select("prediction", "indexedLabel", "indexedFeatures").show()
```

```
+----------+------------+--------------------+
|prediction|indexedLabel|     indexedFeatures|
+----------+------------+--------------------+
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-1....|
|       0.0|         0.0|(4,[0,1,2,3],[0.0...|
|       0.0|         0.0|(4,[0,1,2,3],[0.1...|
|       0.0|         0.0|(4,[0,1,2,3],[0.1...|
|       0.0|         0.0|(4,[0,1,2,3],[0.3...|
|       0.0|         0.0|(4,[0,1,2,3],[0.4...|
|       0.0|         0.0|(4,[0,1,2,3],[0.5...|
|       0.0|         0.0|(4,[0,1,2,3],[0.8...|
|       0.0|         0.0|(4,[0,1,2,3],[1.0...|
|       0.0|         0.0|(4,[0,2,3],[0.166...|
|       0.0|         0.0|(4,[0,2,3],[0.222...|
|       0.0|         0.0|(4,[0,2,3],[0.611...|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
+----------+------------+--------------------+
only showing top 20 rows
```

- Here we import the svm classifier and set the iterations as 2 and then we fit the model with training data.
- Then we set predictions to transform the testing data and print the model stages.
- Then we display the predictions using show().
-



Model Evaluation

You finish codes on the accuracy and precision parts and run the code. Answer the question 4.

F1_score

```
[56] f_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_svm = f_evaluator_svm.evaluate(lsvc_prediction)
     print("f1 score:"+str(f1_score_svm))

     f1 score:0.7337092731829574
```

Recall

```
[57] re_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_svm = re_evaluator_svm.evaluate(lsvc_prediction)
     print("recall:"+str(recall_svm))

     recall:1.0
```

Accuracy

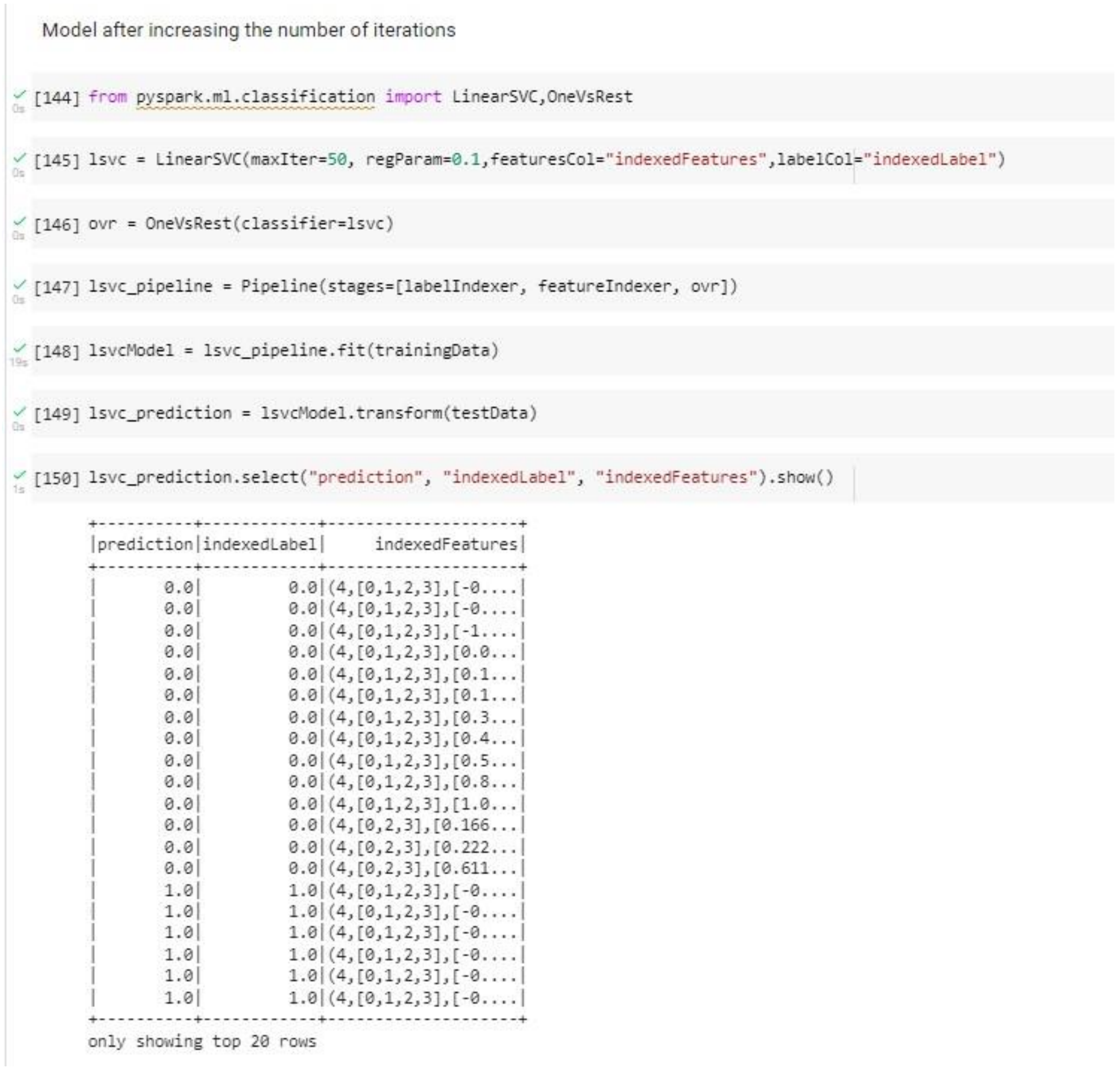
```
[58] acc_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_svm = acc_evaluator_svm.evaluate(lsvc_prediction)
     print("accuracy:"+str(acc_svm))

     accuracy:0.7619047619047619
```

Precision

```
[59] pr_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_svm = pr_evaluator_svm.evaluate(lsvc_prediction)
     print("precision:"+str(precision_svm))

     precision:0.5833333333333334
```

-
- **Here we can check the values**

Model after increasing iterations


Model after increasing the number of iterations

```
[144] from pyspark.ml.classification import LinearSVC,OneVsRest
```

```
[145] lsvc = LinearSVC(maxIter=50, regParam=0.1,featuresCol="indexedFeatures",labelCol="indexedLabel")
```

```
[146] ovr = OneVsRest(classifier=lsvc)
```

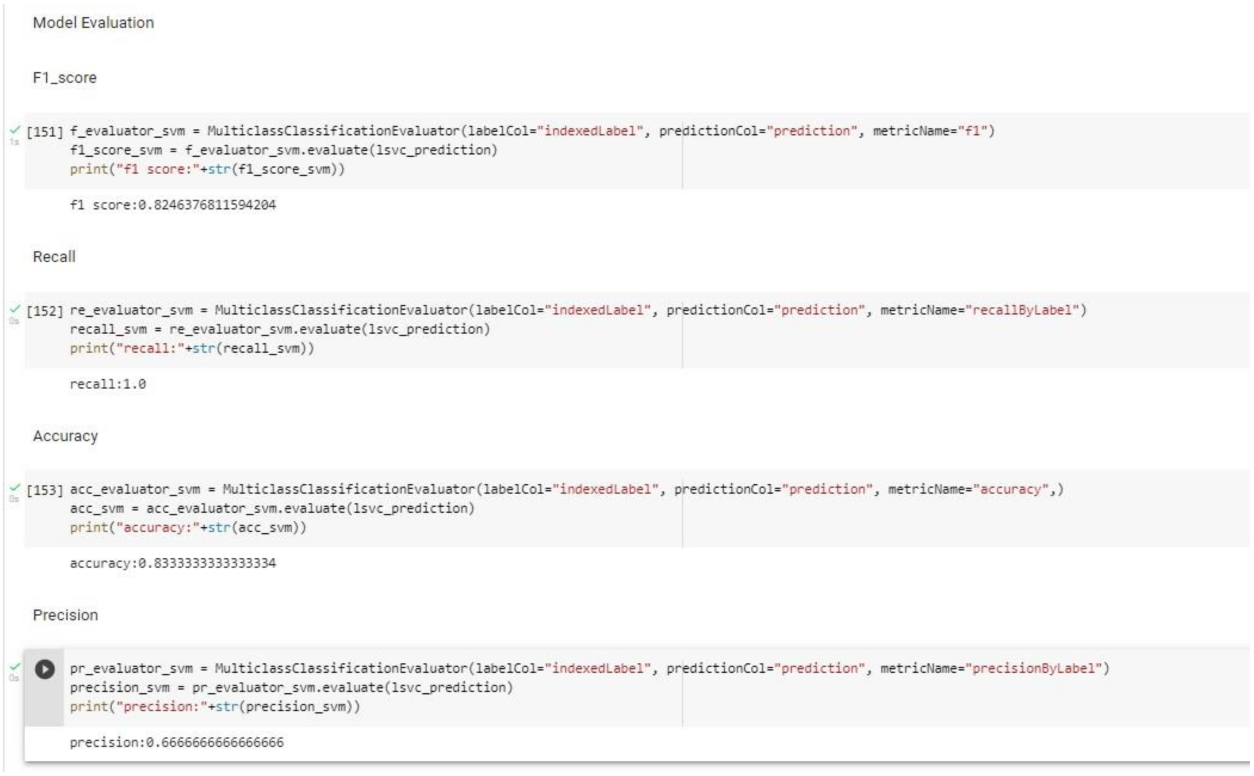
```
[147] lsvc_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, ovr])
```

```
[148] lsvcModel = lsvc_pipeline.fit(trainingData)
```

```
[149] lsvc_prediction = lsvcModel.transform(testData)
```

```
[150] lsvc_prediction.select("prediction", "indexedLabel", "indexedFeatures").show()
```

```
+----------+------------+--------------------+
|prediction|indexedLabel|     indexedFeatures|
+----------+------------+--------------------+
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-0....|
|       0.0|         0.0|(4,[0,1,2,3],[-1....|
|       0.0|         0.0|(4,[0,1,2,3],[0.0...|
|       0.0|         0.0|(4,[0,1,2,3],[0.1...|
|       0.0|         0.0|(4,[0,1,2,3],[0.1...|
|       0.0|         0.0|(4,[0,1,2,3],[0.3...|
|       0.0|         0.0|(4,[0,1,2,3],[0.4...|
|       0.0|         0.0|(4,[0,1,2,3],[0.5...|
|       0.0|         0.0|(4,[0,1,2,3],[0.8...|
|       0.0|         0.0|(4,[0,1,2,3],[1.0...|
|       0.0|         0.0|(4,[0,2,3],[0.166...|
|       0.0|         0.0|(4,[0,2,3],[0.222...|
|       0.0|         0.0|(4,[0,2,3],[0.611...|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
|       1.0|         1.0|(4,[0,1,2,3],[-0....|
+----------+------------+--------------------+
only showing top 20 rows
```



Model Evaluation

F1_score

```
[151] f_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
      f1_score_svm = f_evaluator_svm.evaluate(lsvc_prediction)
      print("f1 score:"+str(f1_score_svm))

      f1 score:0.8246376811594204
```

Recall

```
[152] re_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
      recall_svm = re_evaluator_svm.evaluate(lsvc_prediction)
      print("recall:"+str(recall_svm))

      recall:1.0
```

Accuracy

```
[153] acc_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
      acc_svm = acc_evaluator_svm.evaluate(lsvc_prediction)
      print("accuracy:"+str(acc_svm))

      accuracy:0.8333333333333334
```

Precision

```
      pr_evaluator_svm = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
      precision_svm = pr_evaluator_svm.evaluate(lsvc_prediction)
      print("precision:"+str(precision_svm))

      precision:0.6666666666666666
```

- We increased iterations to 50 and then the performance improved.