【Experiment name】 Prediction of Software Project Popularity Based on Spark MLlib

【Purpose】

Understanding the Background of GitHub Software Project Popularity Predictions

Master the idea of feature engineering, and use SparkSQL for feature extraction

Mastering Spark How to use Pipeline

Mastering Spark How to use common regression models in the M Llib package

【Experimental content】

Topic : Using Spark The MLlib package predicts the popularity of PHP software projects in GitHub, that is, predicts the number of watches for software projects.

Requirements : 1. According to the data set, use SparkSQL to extract as many features as possible for the software project;

2. Divide the test set and training set, build the Spark machine learning pipeline, and use the regression model to predict the popularity of software projects;

3. Evaluate the prediction effect of the model;

4. The prediction results are written to the file system.

Improve (optional) :

1. Try different regression models, try different parameter combinations, and perform hyperparameter tuning.

2. Analyze the GitHub dataset to extract more project features, such as commits.

Experimental results (experimental steps and related core codes) :

1. Import python and spark related packages

2. Fill in the spark configuration information and create a SparkSession

3. Read data

①Set the target folder address to further obtain the target file address

②Set the dataset name ("projects", "issues", "pullrequests")

③ Define the detailed information of a StructField in the schemas

④ Read the target file and create a temporary table

4. Extract data features

① Extract the data characteristics of pullrequest

Among them, the characteristics of Select are

[1] COUNT(head_repo_id),

[2] COUNT(DISTINCT(head_repo_id))

```
[4]: # 提取特征
pullrequest_features = spark.sql("""SELECT base_repo_id,
                                           COUNT(head_repo_id) AS pr_cnt,
                                           COUNT(DISTINCT(head_repo_id)) AS unique_head_repo_cnt
                                    FROM pullrequests
                                    GROUP BY base_repo_id""").cache()
pullrequest_features.createOrReplaceTempView("pullrequest_features")
```

② Extract the data characteristics of the issue

Among them, the characteristics of SELECT are

[1] COUNT(reporter_id)

[2] COUNT(DISTINCT(reporter_id))

[3] COUNT(assignee_id)

```
issue_features = spark.sql("""SELECT project_id,
                                     COUNT(reporter_id) AS issue_cnt,
                                     COUNT(DISTINCT(reporter_id)) AS unique_reporter_cnt,
                                     COUNT(assignee_id) AS assignee_cnt
                              FROM issues
                              GROUP BY project_id""").cache()
issue_features.createOrReplaceTempView("issue_features")
```

5. Synthetic Datasets

① Select the id, label, and features used in the subsequent analysis for synthesis.

②Set the sample to save the synthesis result

```
[5]: # 合成数据集
samples = spark.sql("""SELECT p.id, p.watches_cnt AS label, pr.pr_cnt, pr.unique_head_repo_cnt, i.issue_cnt, i.unique_reporter_cnt, i.assignee_cnt
                       FROM projects AS p
                       LEFT JOIN pullrequest_features AS pr ON pr.base_repo_id=p.id
                       LEFT JOIN issue_features AS i ON i.project_id=p.id""") \
               .fillna(0).cache()
samples.count()
```

6. Start the data analysis process, first import spark related packages

7. Model parameter setting

Set train, test size

```
[25]: train, test = samples.randomSplit([0.8, 0.2], seed=36)  #随机划分训练集，测试集， seed为随机数的种子
```

8. Model calling and training

[26]:
```
va = VectorAssembler(inputCols=["pr_cnt", "unique_head_repo_cnt", "issue_cnt", "unique_reporter_cnt", "assignee_cnt"],
                     outputCol="originalFeatures")
scaler = MinMaxScaler(inputCol="originalFeatures", outputCol="features")  #最大最小值方法，正则化
#lr = LogisticRegression(maxIter=100, regParam=0.001, labelCol="features")
rf = RandomForestRegressor(featuresCol="features")  #随机森林
pipeline = Pipeline(stages=[va, scaler, rf])  #pipeline前3者
model = pipeline.fit(train)
```

[27]:
```
predictions = model.transform(test)  #预测结果
```

9. Model Evaluation

[32]:
```
# 模型评估
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
```

[33]:
```
print(rmse)
```

```
472.61939427239395
```

Attachment: Model Prediction

[34]:
```
predictions.select("id", "label", "prediction").show(20) #显示对应id,label的预测结果
```

```
+--------+-----+------------------+
|      id|label|        prediction|
+--------+-----+------------------+
|   29894|  858|1354.8346777657152|
|  150843|   52| 70.52570795875995|
|  274848|  155| 325.7467649511592|
|  713878|  207| 221.8960967583705|
| 2473361|   39| 258.5021194679377|
| 2875658|  165| 98.52871781368644|
| 4315839|   37|156.73881181104204|
| 4613049|  169|258.32579146361337|
| 6928902|  131| 522.2175529857793|
| 7627296|   87| 71.26670701576725|
| 8089111|   22| 76.34457012406119|
| 9753767|  251|179.00771568785473|
|11117463|   85| 203.6068695565866|
|11516666|  143|120.98104856208492|
|15009179|   57| 98.52871781368644|
|34730394|   85|101.66084408668156|
|56109304|   41|200.67661559047713|
|76962093|   21|103.99875842760875|
|   57513|   34| 98.84529338256941|
|  128423|   31|262.20634701371426|
+--------+-----+------------------+
only showing top 20 rows
```

10. Regression strategy replacement

①Decision tree model for continuous labels

```
rf = DecisionTreeRegressor(featuresCol="features")  #处理连续标签的决策树模型
```

```
# 模型评估
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(rmse)
```

```
519.3993507415956
```

② Gradient boosting tree model for continuous labels

```
rf = GBTRegressor(featuresCol="features")  #处理连续标签的梯度提升树模型
```

```
# 模型评估
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(rmse)
```

```
489.04844786433443
```

③Simple linear regression model

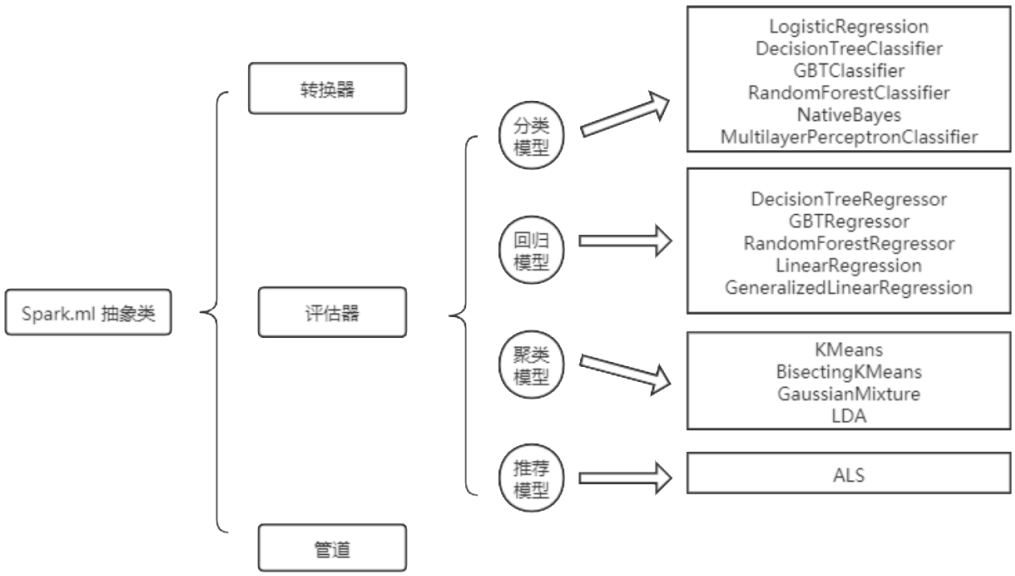
```
rf = LinearRegression(featuresCol="features")  #简单线性回归模型
```

```
# 模型评估
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(rmse)
```

```
444.2407621697441
```

【Experiment Summary】

Through the study of this experimental course, I understand the background of GitHub software project popularity prediction, master the idea of feature engineering, and use SparkSQL for feature extraction, Spark How to use Pipeline with Spark How to use common regression models in the M Llib package. In particular, the method of model invocation and evaluation is learned. In the example given by the teacher, we learned and used the maximum minimum method and the random forest method for regression prediction, and finally evaluated the model. Under the regression framework of the same s park.ml , I also tried and called the decision tree model for continuous labels , the gradient boosting tree model for continuous labels, and the simple linear regression model, and evaluated the results. This experiment gave me a good new understanding of s park in machine learning, especially regression prediction, and a better grasp of s park .



For example, the picture above shows the existing classification, regression, clustering and recommendation models in s park.ml. I believe that through continuous experiments in the future, I will have a deeper understanding of these models.