FIT5202 Data processing for big data¶

Activity: Machine Learning with Spark (Model Selection and Persistance)¶

In this part of the tutorial, we want to look into 2 things 1) Model Selection and 2)Persisting the Model. The sequence of steps below create the ML Pipeline for the Decision Tree Model that we did in Week 6.

In [ ]:

```
#Write your code here
from pyspark import SparkConf # Spark
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml import Pipeline

spark_conf = SparkConf()\
             .setMaster("local[*]")\
             .setAppName("ML-Classification")

spark = SparkSession.builder.config(conf=spark_conf).getOrCreate()
spark.sparkContext.setLogLevel("ERROR")

df = spark.read.csv('bank.csv', header = True, inferSchema = True)
cols = df.columns

# First, save the category in the category columns list.
categoryInputCols = ['job', 'marital', 'education', 'default', 'housing', 'loan',
'contact', 'poutcome']
numericInputCols = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']
categoryOutputCol = 'deposit'
categoryCols = categoryInputCols+[categoryOutputCol]

train, test = df.randomSplit([0.7, 0.3], seed = 2020)

outputCols=[f'{x}_index' for x in categoryInputCols]
outputCols.append('label')
inputIndexer = StringIndexer(inputCols=categoryCols, outputCols=outputCols)

inputCols_OHE = [x for x in outputCols if x!='label']
outputCols_OHE = [f'{x}_vec' for x in categoryInputCols]
encoder = OneHotEncoder(inputCols=inputCols_OHE,outputCols=outputCols_OHE)

inputCols=outputCols_OHE
assemblerInputs = outputCols_OHE + numericInputCols
assembler = VectorAssembler(inputCols = assemblerInputs, outputCol="features")

dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth =
3)

stage_1 = inputIndexer
stage_2 = encoder
stage_3 = assembler
```

```
stage_4 = dt

stages = [stage_1,stage_2,stage_3,stage_4]

pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(train)
predictions = pipelineModel.transform(test)
```

In [ ]:

```
predictions.select('features','label','prediction').show()
```

Accessing the parameters in the Model¶

You can use `extractParamMap()` to see the list of parameters for the particular estimater. For more details on this, refer to the Spark Documentation. If it is a PipelineModel, you need to do `model.stages[-1].extractParamMap()`

In [ ]:

```
pipelineModel.stages[-1].extractParamMap()
```

Cross Validation and Hyperparameter Tuning¶

Last week we looked into Decision Trees, out of the different parameters, let's take `maxBins` and `maxDepth` as the two hyperparamters. We used `maxDepth=3` as the default value, but **is 3 the optimum hyperparameter value for the model?**. This is something we want to achieve using the HyperParameter Tuning. We could also manually tune the hyperparameters using educated guesses, training the model, evaluating its performance and repeating the steps but it will be tedious. Read More

One popular approach is to create a grid of hyper-parameters we want to optimze with the values we want to try out. In Spark, we can use `ParamGridBuilder` to define the hyperparameters for any estimator. Since the model needs to be evaluated at every parameter combination, we need to also define an `Evaluator`.

Finally, when we use this with the `CrossValidator (K-Fold)`, for each fold (i.e. the train/test split), it tries out all the possible combination of the hyper-parameters and evaluates the performance of each instance. Finally, based on the evaluaton, it gives us the best model i.e. the best combination of hyperparameters to use.

Let's try to tune the parameters for our `DecisionTree` Model. Since we are using the `Pipeline`, we can directly plugin the PipelineModel to the CrossValidator as well.

Let's build a grid specifying all the parameters and their values we want to test our model with. We are assigning a series of values for the hyperparameters `maxDepth` and `maxBins`.

In [ ]:

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator,CrossValidatorModel
from pyspark.ml.evaluation import BinaryClassificationEvaluator
# Create ParamGrid for Cross Validation
dtparamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [2, 5, 10, 20, 30])
             .addGrid(dt.maxBins, [10, 20, 40, 80, 100])
             .build())
```

In [ ]:

```
# Define an evaluator to be used for evaluating the model
dtevaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
```

Finally, let's declare the `CrossValidator` which takes the estimator, paramgrid and evaluator as input. Also, we need to specify the number of folds we want to test against.

In [ ]:

```
# Create 3-fold CrossValidator
dtcv = CrossValidator(estimator = pipeline,
```

```
                    estimatorParamMaps = dtparamGrid,
                    evaluator = dtevaluator,
                    numFolds = 3)
```

This is where we train our cross-validator, as the CV trains/evaluates the model for every fold of data across all possible parameter combinations, **this step is very expensive and might take some time to finsh.**

In [ ]:

```
# Run cross validations
dtcvModel = dtcv.fit(train)
print(dtcvModel)
```

Finding the Best Model¶

Now that we have finished running the CrossValidator, we can obtain the model with the best combination of hyperparameters using `.bestModel`. Also `bestModel.stages[-1]._java_obj.paramMap()` gives you the hyperparameteres with the optimum values selected by the CV.

In [ ]:

```
#Getting the best model and the optimum parameters selected from the Cross Validation
bestModel= dtcvModel.bestModel
print(bestModel.stages)
print('Best Param for DT: ', bestModel.stages[-1]._java_obj.paramMap())
```

Model Persistance (Saving and Loading the Model)¶

For simple models (i.e. without pipelines), you can simply save the model by using `model.save('path')` and load it using `.load('model_path')`.

You can also save and load whole PipelineModel in Spark using save/load methods. In the following example, we will save the **bestModel** we obtained from the **Model Selection** and Load it again.

In [ ]:

```
#Saves the model to the filesystem
bestModel.save('bank_subscriber_prediction_model')
```


In [ ]:

```
#Loading the Pipeline Model From the filesystem
from pyspark.ml import PipelineModel
pipelineModel = PipelineModel.load('bank_subscriber_prediction_model')
```

The above step loads the same model again, you can check the hyperparameters that we obtained earlier for this model.

In [ ]:

```
print(pipelineModel.stages[-1]._java_obj.paramMap())
```

TODO: You can load this model in a separate file and try to generate some predictions off it directly by using some data instances from the bank.csv file.

TrainValidation Split¶

This is another aproach in Spark for hyper-parameter tuning. You can refer to the Spark Documentation for more details [Ref]. Compared to CrossValidation, this approach is less expensive since it evaluates each combination of parameters just once as opposed to k-times in CV. The example below demonstrates the use of TrainValidationSplit. We have used the same parameter combination with the same pipeline model and evaluator.

Note that, one input parameter that is different than CV is `trainRatio`, which specifies the split percentage for train/validation data. When you run this vs the Cross-Validation version, you will notice signficant difference in the time taken which is due to the fact that this approach only

evaluates the combination of parameters once.

In [ ]:
```
from pyspark.ml.tuning import TrainValidationSplit
dtparamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [2, 5, 10, 20, 30])
             .addGrid(dt.maxBins, [10, 20, 40, 80, 100])
             .build())

dttv = TrainValidationSplit(estimator = pipeline,
                      estimatorParamMaps = dtparamGrid,
                      evaluator = dtevaluator,
                      trainRatio = 0.8)
model = dttv.fit(train)
```

In [ ]:
```
bestModel_tv = model.bestModel
```

In [ ]:
```
print(bestModel_tv.stages[-1]._java_obj.paramMap())
```

References¶

1. Machine Learning Model Selection and Hyperparameter Tuning using PySpark
2. Spark ML Tuning