Grid Dynamics

Lecture 3: Custom Transformers and Estimators

Lecture 3: Custom Transformers and Estimators

- Introduction
- About Spark
- Spark MLlib
- Classification
  - Naive Bayes Classifier
  - Logistic Regression
  - Decision Tree Classifier
  - Random Forest Classifier
  - GBT (Gradient-Boosted Trees) Classifier
  - Multilayer Perceptron Classifier
- Regression
  - Linear Regression
- Clustering
  - K-Means
- Collaborative filtering
  - ALS (alternating least squares)
- Frequent Pattern Mining
  - FPG (Frequent Pattern-Growth)
- Custom Transformer
- Custom Estimator
- Demo
- Summary
- QA

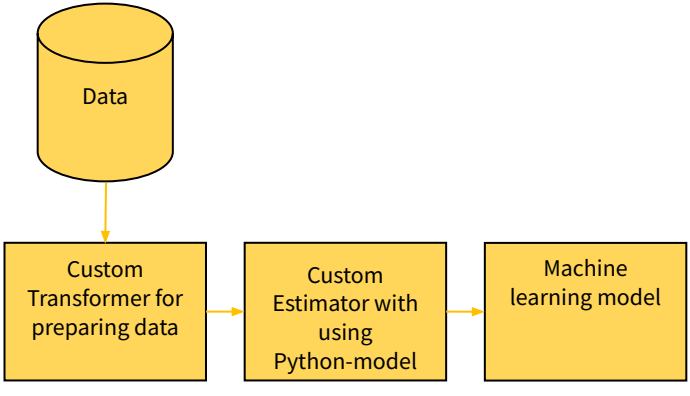
Test case

We have an online cinema.

We have various indicators - how many hours users are online, what movies they rate, how many promo codes they used, etc.

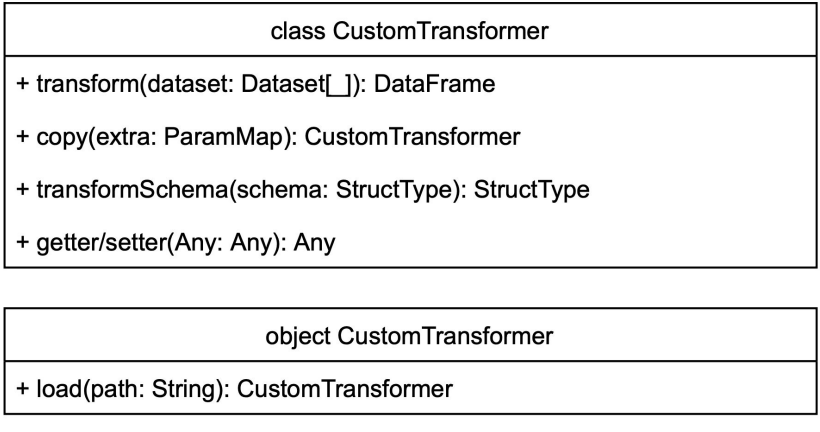
We want to predict how much users spend per month.

We need special data converters. The machine learning model is implemented in python and it needs to be integrated into Scala code.



Custom Transformer

Custom Transformer



Custom Transformer

```
class CustomTransformer(override val uid: String) extends Transformer with DefaultParamsWritable {
  def this() = this(Identifiable.randomUID("org.apache.spark.ml.feature.CustomTransformer"))


  override def transform(dataset: Dataset[_]): DataFrame = {
    ??? // Necessary transformations,
        // e.g.: dataset.withColumn("len", length($"col"))
  }


  override def copy(extra: ParamMap): CustomTransformer = defaultCopy(extra)

  override def transformSchema(schema: StructType): StructType = {
    ??? // Defining the output data schema, e.g.:
        // StructType(Seq(StructField("col",StringType,true),
        // StructField("len",IntegerType,true)))
   }
  }


  object CustomTransformer extends DefaultParamsReadable[CustomTransformer] {
    override def load(path: String): CustomTransformer = super.load(path)
  }
```

Custom Estimator

class CustomEstimator
+ fit(dataset: Dataset[_]): CustomEstimatorModel
+ copy(extra: ParamMap): CustomEstimator
+ transformSchema(schema: StructType): StructType
+ getter/setter(Any: Any): Any

object CustomEstimator
+ load(path: String): CustomEstimator = super.load(path)

class CustomEstimatorModel
+ transform(dataset: Dataset[_]): DataFrame
+ copy(extra: ParamMap): CustomEstimator
+ transformSchema(schema: StructType): StructType
+ getter/setter(Any: Any): Any

object CustomEstimatorModel
+ read: MLReader[CustomEstimatorModel]
+ load(path: String): CustomEstimatorModel
+ class CustomEstimatorModelWriter
+ class CustomEstimatorModelReader

Custom Estimator

```
class CustomEstimator(override val uid: String) extends Estimator[CustomEstimatorModel] with
DefaultParamsWritable {
  def this() = this(Identifiable.randomUID("org.apache.spark.ml.feature.CustomEstimator"))


  override def fit(dataset: Dataset[_]): CustomEstimatorModel = {
    ??? // model training, can be trained using python through
        // rdd.pipe(filename), e.g.:
        // val rddModel = dataset.rdd.pipe(s"train.py")
        // val text = rddModel.toDF().select(modelDF("value")).collect
        // val textStr = text(0)(0).asInstanceOf[String]
        // new CustomEstimatorModel(uid, textStr)
  }


  override def copy(extra: ParamMap): CustomEstimator = defaultCopy(extra)
  override def transformSchema(schema: StructType): StructType = {
    ??? // Defining the output data schema, e.g.:
        // StructType(Seq(StructField("preds",DoubleType,true)))
  }
}

object CustomEstimator extends DefaultParamsReadable[CustomEstimator] {
  override def load(path: String): CustomEstimator = super.load(path)
}
```

train.py

```
rows = [] # init empty rows array for DataFrame
for line in sys.stdin:
    # parsing RDD
    # create pandas DataFrame
df = pd.DataFrame(rows)

# init features columns and label column
feature_columns = [“feature1”, “feature2”]
label_column = "label"

#fit model (e.g. LogisticRegression)
model = LogisticRegression()
model.fit(df[feature_columns], df[label_column])

# convert model to base64 string
model_string = base64.b64encode(pickle.dumps(model)).decode('utf-8')
# return model
print(model_string)
```

Custom Estimator

```
class CustomEstimatorModel(override val uid: String, val model: String)
  extends Model[CustomEstimatorModel] with MLWritable {
  // “model” in input args is a training model from python
  override def copy(extra: ParamMap): CustomEstimatorModel = defaultCopy(extra)

  override def transform(dataset: Dataset[_]): DataFrame = {
    ??? // Using the previously trained model, e.g.:
        // new BufferedWriter(new FileWriter(new
        // File(“lregres.model"))).write(model)
        // dataset.sparkSession.sparkContext.addFile("lregres.model", true)
        // val rddRes = df.rdd.pipe(s"./test.py")
        // var res = rddRes.toDF()
        // val split_col = split(res("value"), "[ ]+")
        // res.withColumn("preds", split_col.getItem(1))
  }


  override def transformSchema(schema: StructType): StructType = {
    ??? // Defining the output data schema, e.g.:
        // StructType(Seq(StructField("preds",DoubleType,true)))
  }
  override def write: MLWriter = new CustomEstimatorModelWriter(this)
}
```

test.py

```
# read the model, deserialize and unpickle it.

model = pickle.loads(base64.b64decode(open("lregres.model").read().encode('utf-8')))

rows = [] #here we keep input data to Dataframe constructor

# iterate over standard input
for line in sys.stdin:
    #parse line into a dict: {"column1": value1, ...}

#initialize a dataframe from the list
df = pd.DataFrame(rows)

#run inference
pred = model.predict(df)

# Output to stdin, so that rdd.pipe() can return the strings to pipedRdd
print(pred)
```

Custom Estimator

```
object CustomEstimatorModel extends MLReadable[CustomEstimatorModel] {
  private[CustomEstimatorModel]

  override def load(path: String): CustomEstimatorModel = super.load(path)
  override def read: MLReader[CustomEstimatorModel] = new
CustomEstimatorModelReader
  class CustomEstimatorModelWriter(instance: CustomEstimatorModel) extends
MLWriter {
    private case class Data(model: String)

    override protected def saveImpl(path: String): Unit = {
      DefaultParamsWriter.saveMetadata(instance, path, sc)
      val data = Data(instance.model)
      val dataPath = new Path(path, "data").toString
      sparkSession.createDataFrame(Seq(data))
                .repartition(1).write.parquet(dataPath)
    }
}
```

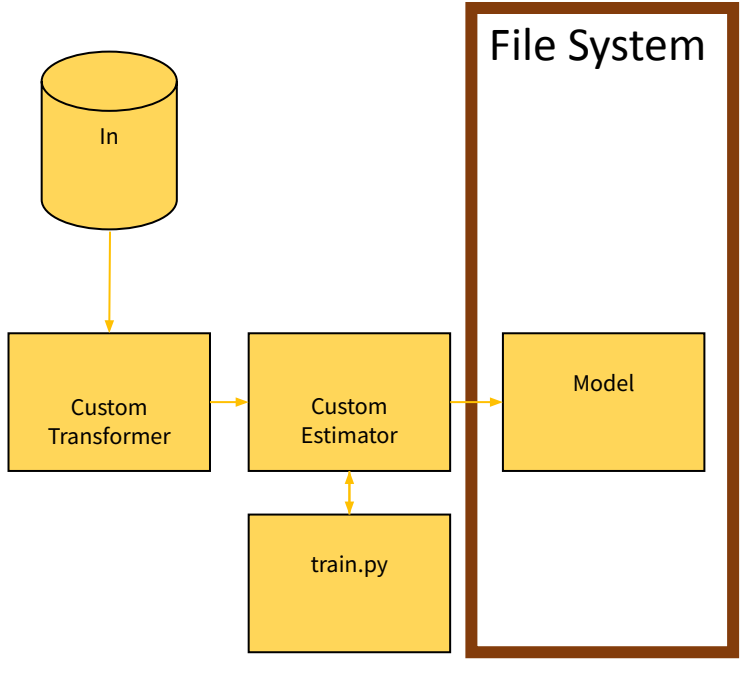
Custom Estimator

```
private class CustomEstimatorModelReader extends MLReader[CustomEstimatorModel] {
       private val className = classOf[CustomEstimatorModel].getName

       override def load(path: String): CustomEstimatorModel = {
            val metadata = DefaultParamsReader.loadMetadata(path, sc, className)
            val dataPath = new Path(path, "data").toString
            val data = sparkSession.read.parquet(dataPath)
                           .select("model")
                           .head()
            val modelStr = data.getAs[String](0)
            val model = new CustomEstimatorModel(metadata.uid, modelStr)
            metadata.getAndSetParams(model)
            model
        }
}
}
```

Pipeline

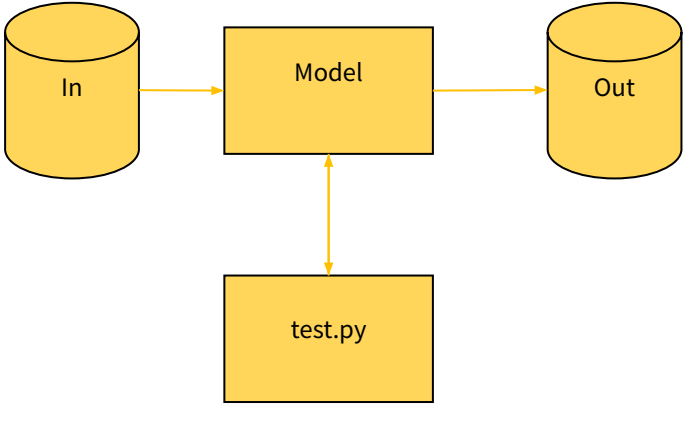
Train Pipeline

```
val df =
spark.read.format("json")
  .json(trainPath)

val transformer =
  new CustomTransformer()

val ce = new CustomEstimator()

val pipeline = new Pipeline()
  .setStages(
    Array(transformer, ce))

val model = pipeline.fit(df)

model.write.overwrite()
  .save(modelPath)
```



Test Pipeline

```
val df =
spark.read.format("json
")
  .json(testPath)

val model =
PipelineModel
  .load(modelPath)

var res =
model.transform(df)
```



Task 3. Classification model with custom Transformers and Estimators. Identification of survivors based on data on passengers of the Titanic.

The task is to train a machine learning model that will predict whether the passenger survived the crash or not based on the presented data.

Conditions:

- Model should be trained using custom extensions of the Spark ML transformer and estimator classes and integration of Python libraries. You can use the skeleton code prepared for you in SparkCustomMLPipeline folder.
- You need to train the model on a training data set `train.csv`.
- You need to test the model on a test dataset `test.csv`.
- Calculate metrics based on the file `is_survived.csv`.
- Compare the results of two models.

Summary

Got acquainted with the Spark MLlib library, which allows you to build a Data processing Pipeline

Considered the possibility of creating custom transformers and estimators based on Spark base classes.

Looked at examples of how this can be implemented. In the example, considered the possibility of integrating several languages.

Q&A

Your feedback & questions

Vmonastyrev@griddynamics.com

github: https://github.com/MrSandmanRUS

Telegram: @VitaliyMonastyrev

Vitalii Monastyrev

isamsonov@griddynamics.com

github: https://github.com/prinstonsam

Telegram: @prinstonsam

Igor Samsonov

Thank you!

Grid Dynamics International, Inc.

500 Executive Parkway,
Suite 520 / San Ramon, CA
+1 650-523-5000
info@griddynamics.com
www.griddynamics.com