Coursework Distributed Databases

Wout Boeykens [1], Hannes Roegiers [2], Strijk Philippe [3]

Samenvatting
This task discusses 3 Kaggle competitions

Sleutelwoorden
Apache — Spark — Maven — Big Data — Machine Learning — Classification — Regression — Kaggle — Distributed Databases

Co-promotor
[2] (Stijn Lievens)

Contact: [1] wout.boeykens@student.hogent.be; [2] hannes.roegiers@student.hogent.be; [3] philippe.strijk@student.hogent.be

Inhoudsopgave

1 Introductie 1

2 Initialisatie 1

3 Een kijkje nemen 1

3.1 Quora Incinsere Question Classification 2
Hoe zien de datasets eruit? • Hoe groot zijn de sets? • Hoe zijn de targets verdeeld?

4 Data Preprocessing 2

4.1 Quora Incinsere Question Classification 2
Tokenization • Cleaning up • Hashing and IDF'ing

5 Model Selection 3

5.1 Quora Incinsere Question Classification 3
Logistic Regression

6 Conclusions 3

6.1 Quora Incinsere Question Classification 3

6.2 Tabular Playground Series - Nov 2022 3
Spark vs Python • werken met ML-lib • results

6.3 House Prices - Advanced Regression Techniques 3
Spark vs Python • problemen

1. Introductie

Bij aanvang van deze taak was onze kennis over Java Spark en zijn Machine Learning implementaties beperkt, als niet onbestaande. Code vertalen van makkelijk bruikbare python scikit-learn naar Java Apache Spark libraries, bleek een uitdaging te zijn die ons door meerdere konijnhollen nam. Het unieke karakter van deze taak was evident vanaf het begin. Deze taak bespreekt 3 Kaggle competities, namelijk de 'Quora Insincere Question Classification, House Prices - Advanced Regression Techniques, 'Tabular Playground Series - Nov 2022'. De paper zal per hoofdstuk de algemene takeaways toelichten, en vervolgens per competitie specifiek de gedachtengang uitleggen.

2. Initialisatie

Allereerst volgen enkele algemene initalisaties: het random seed getal, de sparksessie, een logger, en de datasets om in te lezen.

Random seed getal:

```
    private static final int SEED = 42;
```

Spark sessie:

```
// Create a Spark Session
    SparkSession spark = SparkSession.
        builder()
    .appName("Spark Demo")
    .master("local[*]")
    .config("spark.logLineage", true)
    .getOrCreate();
```

Logger:

```
// Create a Logger
    Logger logger = Logger.getLogger("org.
        apache");
    logger.setLevel(Level.WARN);
```

Datasets inlezen:

```
// Read in files: test set, train set
    Dataset<Row> testSet = spark.read()
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("src/main/resources/test.csv");

    Dataset<Row> trainSet = spark.read()
    .option("header", "true")
    .option("inferSchema", "true")
    .csv("src/main/resources/train.csv");
```

3. Een kijkje nemen

Voordat de data wordt aangepast en het model geselecteerd wordt, moet de data begrepen en bekeken worden.

3.1 Quora Incinsere Question Classification

3.1.1 Hoe zien de datasets eruit?

Test set:

```
+--------------------+--------------------+
|                 qid|       question_text|
+--------------------+--------------------+
|0000163e3ea7c7a74cd7|Why do so many wo...|
|00002bd4fb5d505b9161|When should I app...|
|00007756b4a147d2b0b3|What is it really...|
+--------------------+--------------------+
```

In het geval van de test set is dit goed; er is een unieke identifier en een string aan teksten, geen labels.

Train set:

```
+--------------------+--------------------+------+
|                 qid|       question_text|target|
+--------------------+--------------------+------+
|00002165364db923c7e6|How did Quebec na...|     0|
|000032939017120e6e44|Do you have an ad...|     0|
|0000412ca6e4628ce2cf|Why does velocity...|     0|
+--------------------+--------------------+------+
```

Hetzelfde als de test set, maar wel met labels.

3.1.2 Hoe groot zijn de sets?

Wat is de grootte van de sets?

```
Size test set: 375810   Size train set: 1306140
```

3.1.3 Hoe zijn de targets verdeeld?

Hoeveel vragen hebben we in de training data die gelabeled zijn als 'incinsere', met een 1?

```
    Size with target = 1: 79807
```

We kunnen hieruit concluderen dat uit de training set van 1.306.140 entries, er 79.807 gelabeled zijn als 'incinsere'. Dit is 6.11% van de gevallen.

4. Data Preprocessing

4.1 Quora Incinsere Question Classification

4.1.1 Tokenization

Allereerst moeten alle woorden getokenized worden.

```
Tokenizer tokenizer = new Tokenizer().
    setInputCol("question_text").
    setOutputCol("tokens");
trainSet = tokenizer.transform(trainSet);
```

4.1.2 Cleaning up

Specifieke waarden verwijderen uit een dataset in Java Spark kan makkelijk gedaan worden via UDF's. Deze UDF gaan we gebruiken om komma's, vraagtekens en punten te verwijderen. Qua redenering komt deze stap na de tokenization, het is echter makkelijker om de UDF toe te passen op String objecten, vandaar zetten we hem in de code vóór de tokenization.

```
UDF1<String, String> replaceQM = new
    UDF1<String, String>() {
    public String call (final String str)
        throws Exception {
            return str.replaceAll("
                [\\?,.]", "");
        }
    };
// Register the UDF
spark.udf().register("replaceQM",
    replaceQM, DataTypes.StringType);
// Apply the UDF
trainSet.select(functions.callUDF("
    replaceQM", trainSet.col("
    question_text"))
.alias("column_name_without_QM"));

trainSet = trainSet.withColumn("
    column_name_without_QM",
    functions.callUDF("replaceQM",
    trainSet.col("question_text")));
trainSet = trainSet.drop("
    question_text").withColumnRenamed
    ("column_name_without_QM", "
    question_text");
```

Vervolgens worden de stopwoorden die heel vaak voorkomen, maar weinig zeggen over de inhoud, verwijderd. Na het bekijken van de dataset, zijn er een aantal woorden die men extra verwijderd wilt zien. Zo heeft de StopWordsRemover daar een setter voor. Om de standaard woordenlijst te behouden, en er nieuwe aan toe te voegen, halen we de standaard woordenlijst op, en voegen samen met degene die we zelf gedeclareerd hebben. De woordenlijst die hier gedeclareerd wordt is op basis van een Kaggle notebook.

```
StopWordsRemover remover = new
    StopWordsRemover()
    .setInputCol("tokens")
    .setOutputCol("tokens w/o stopwords"
        );
    String[] originalStopwords = remover
        .getStopWords();
    String[] stopwords = new String[] {"
        one ", "br", "Po", "th", "sayi",
        "fo", "unknown"};
    String[] allStopwords =
        combineObjects(stopwords,
        originalStopwords);
    remover.setStopWords(allStopwords);
    trainSet = remover.transform(
        trainSet);
```

De datasuit die hieruit volgt ziet er zo uit:

```
+--------------------+------+--------------------+
|                 qid|target|              tokens|
+--------------------+------+--------------------+
|00002165364db923c7e6|     0|[quebec, national...|
|000032939017120e6e44|     0|[adopted, dog, en...|
|0000412ca6e4628ce2cf|     0|[velocity, affect...|
|000042bf85aa498cd78e|     0|[otto, von, gueri...|
+--------------------+------+--------------------+
```

Dit is het soort dataset dat we willen zien. Geen stopwoorden of leestekens.

4.1.3 Hashing and IDF'ing

Vervolgens moeten de woorden gehashed worden:

```
HashingTF hashingTF = new HashingTF().
    setInputCol("words").setOutputCol("
    rawFeatures");
trainSet = hashingTF.transform(trainSet)
    ;
```

Wat resulteert in de 'rawFeatures' kolom:

```
+--------------------+------+--------------------+
|                 qid|target|         rawFeatures|
+--------------------+------+--------------------+
|00002165364db923c7e6|     0|(262144,[8538,374...|
|000032939017120e6e44|     0|(262144,[54556,12...|
|0000412ca6e4628ce2cf|     0|(262144,[27783,29...|
+--------------------+------+--------------------+
```

Nu worden de woorden gescaled op basis van een IDF model.

```
IDF idf = new IDF().setInputCol("
    rawFeatures").setOutputCol("features"
    );
IDFModel idfTrainModel = idf.fit(
    trainSet);
trainSet = idfTrainModel.transform(
    trainSet);
```

5. Model Selection

5.1 Quora Incinsere Question Classification

Voor dit vraagstuk, zal (naïef) een lineair regressiemodel toegepast worden.

5.1.1 Logistic Regression

Het model wordt als volgt gedefinieert:

```
LogisticRegression mlr = new
    LogisticRegression()
       .setMaxIter(10)
       .setRegParam(0.3)
       .setElasticNetParam(0.8);

  LogisticRegressionModel mlryTrainModel =
       mlr.fit(trainSet);
  Dataset<Row> predictions =
      mlryTrainModel.transform(testSet);
```

Helaas crasht het programma hier met een OutOfMemoryError: Java Heap Space

We ommzeilen dit door één vijfde van de dataset te gebruiken.

```
Dataset<Row>[] split = trainSet.
    randomSplit(new double[]{0.8, 0.2},
    SEED);
trainSet = split[1];
```

```
LogisticRegressionModel mlryTrainModel =
    mlr.fit(trainSet);
Dataset<Row> predictions =
    mlryTrainModel.transform(testSet);
```

6. Conclusions

6.1 Quora Incinsere Question Classification

Het model bereikte een accuraatheid van .93816% Op het eerste blik, lijkt dit een geweldig resultaat voor het voorspellen van of een vraag oprecht is of niet. Maar na een logische kijk op de zaken, blijkt duidelijk dat de accuraatheid van 0.938% voortkomt uit de 6.11% van de test set die een 1 hebben gescoord. Met andere woorden, het model voorspelt elke woordcombinatie in de test set als een 0. Het model onderfit de data, een andere aanpak zal nodig zijn. Deze Kaggle competitie is niet opgelost en schijnt een probleem te zijn waarbij gesofisticeerdere methodes dan lineaire regressie gebruikt moeten worden. In deze taak heb ik ook Random Forest gebruikt, maar die kreeg bij het trainen van het model een Java Out of Heap Error, zelfs bij een train set van 2000 entries.

6.2 Tabular Playground Series - Nov 2022

6.2.1 Spark vs Python

Ik heb het gevoel dat python je iets meer helpt en er meer al voor jou gedaan wordt. Maar beide hebben redelijk dezelfde functionaliteiten. Waar ik bij Spark wel veel moeite mee had was de feature engineering. In python ging dit iets vlotter, ook de eerste keren dat ik er mee werktte.

6.2.2 werken met ML-lib

Ik vind met python werken sowieso al makkelijker dan met java te werken. Maar ik had niet het gevoel dat het super moeilijk was om met ML-lib te werken, dit kan natuurlijk ook zijn omdat ik veel dingen herkende vanuit python. De code zelf vond ik ook iets overzichtelijker dan python code.

6.2.3 results

Het lukte me niet om een goed resultaat te bekomen omdat ik telkens een exception kreeg wanneer ik .fit toepaste op mijn Pipeline. Het probleem is dat mijn data nog niet gevectorized is, ik heb dit lang proberen oplossen maar slaagde daar niet in. Als ik nog eens een opdracht heb waar ik met een nieuwe library moet werken, ga ik meer de tijd nemen om eerst alles wat beter te begrijpen voor ik begin aan het programmeer gedeelte zelf. Want tijdens deze opdracht ontdekte ik vaak andere manieren om iets te doen die vaak beter zijn dan de manier waarop ik het deed.

6.3 House Prices - Advanced Regression Techniques

6.3.1 Spark vs Python

Python maakt het veel gemakkelijker als je classificaties wilt omzetten in vectors voor een ML model. Python heeft ingebouwde functies die gemakkelijk omgaan met verschillende datatypes, wat handig is voor grote dataframes.

6.3.2 problemen

Ik heb de grootste problemen ondervonden met het mappen van de kolommen met categorieën. Aangezien sommige kolommen getallen waren en ander categorieën moesten deze opgesplitst worden zodat ze op de juiste manier getransformeerd werden.

Een ander probleem was een OutOfMemoryEror: Java Heap Size. Dit kon ik oplossen door een VM argument mee te geven met eclipse '-Xmx1024m'. Dit zette de heap size op 1GB.

Het laatste probleem dat ik ondervonden heb was een probleem met de versie van MLlib van Spark. Ik heb de pom-file aangepast naar een versie ouder is dan 2.13.

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.12</
        artifactId>
    <version>3.3.1</version>
    <scope>provided</scope>
</dependency>
```