Classificação de Avaliações de Produtos da Amazon

B. Kimura, E. Malzoni.

1 INTRODUÇÃO

Esse projeto tem como objetivo estudar como se comporta as avaliações de produtos no site da Amazon. Para tanto, foi feita uma análise para entender como o comentário associado a avaliação influencia a nota atribuída ao produto. Com isso, é possível determinar a nota que um cliente daria a um produto, apenas olhando o comentário feito por ele.

2 PROCEDIMENTO

2.1 Processamento de dados

Os dados utilizados para fazer a análise, foram retirado do próprio site da Amazon[1], armazenado no s3. O dataset escolhido foi sobre os dados de *reviews* de produtos vendidos na Amazon apenas nos Estados Unidos da América e da França[2]. Como ferramenta para o estudo foi usado o *pyspark* no *zeppelin*.

Para abrir os dados há duas formas diferentes. A primeira possibilidade é usar uma função do próprio *spark*, como mostra a Figura 1. Já uma segunda forma, a qual decidimos utilizar, foi abrir com uma biblioteca que disponibiliza os dados em formato SQL. Assim, é possível programar utilizando as função nessa linguagem. A Figura 2, ilustra como ficou o código.

```
df = spark.read.load("s3://amazon-reviews-pds/parquet/")
```

Figura 1 - Leitura dos dados com a biblioteca spark

```
sqlContext = SQLContext(sc)
data = sqlContext.read.load("s3://amazon-reviews-pds/parquet/")
```

Figura 2 - Leitura dos dados com a biblioteca sqlContext

2.2 Geração do treinamento

O treinamento para determinar qual a nota dada a um determinado comentário foi baseado no código de Susan Li, o qual explica como fazer classificação de texto utilizando o *pyspark*[3]. Além disso, utilizamos a documentação da biblioteca *ml* do *pyspark*[4].

O primeiro passo foi transformar a categoria "*star_rating*" que possui cinco valores (1, 2, 3, 4 e 5 estrelas) em uma categoria binária, ou seja, para notas abaixo de 3 o valor seria 0 e para valores acima disso a nota seria 1, significando que as avaliações possuem uma conotação negativas e positivas respectivamente (Figura 3).

```
targetDf = data.withColumn("star_rating", when(data["star_rating"] < 3, 0).otherwise(1))
```

Figura 3 - Transformação da categoria "*star_rating*" para binária

Como os dados de resenha são extremamente grandes, foi necessário diminuir o tamanho da amostra, segundo a Figura 4. Para que não seja necessário executar desnecessariamente várias vezes o código demonstrado pela Figura abaixo, foi utilizado o "*cache*", possibilitando que o valor do *data_sample* fique armazenado na memória.

```
data_sample = data.sample(withReplacement=False, fraction=0.00001, seed=0).cache()
```

Figura 4 - Redução do tamanho da amostra

Com as categorias definidas o próximo passo foi "*tokenizar*" o texto, ou seja, separar o texto em palavras. Desse agrupamento de palavras foi retirada as palavras que não possuíam valores positivos ou negativos (*stop_words*) como o

"the" (o), por exemplo, que é apenas um artigo e pertence tanto a resenhas positivas quanto negativas. As palavras restantes foram colocadas em um vetor que possui a contagem de cada uma dessas palavras. Para tabela final, além das colunas geradas com o código até então ("*star_rating*", "*review_body*", "*words*", "*filtered*", "*features*"), foi adicionada mais uma coluna denominada "*label*". Como mostra a Figura 5. O cabeçalho dessa tabela está demonstrada na Figura 6, onde "*star_rating*" representa se a resenha é positiva o negativa, "*review_body*" o texto da avaliação, "*words*" as palavras após a "*tokenização*", "filtered" as palavras com a remoção das "*stop words*", "*features*" os desritores das *reviews*, e por fim, "label" que possui o valor inverso da "star_rating" nesse caso.

```
regexTokenizer = RegexTokenizer(inputCol="review_body", outputCol="words", pattern="\\W")

add_stopwords = ["the"]
stopwordsRemover = StopWordsRemover(inputCol="words",
outputCol="filtered").setStopWords(add_stopwords)

countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=10000,
minDF=1)

label_stringIdx = StringIndexer(inputCol = "star_rating", outputCol = "label")

pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label_stringIdx])
```

Figura 5 - Código de *tokenização,* filtragem e criação do pipeline

```
+--------------+--------------+---------+------------+------------+---------+
|  star_rating  |  review_body  |  words  |  filtered  |  features  |  label  |
+--------------+--------------+---------+------------+------------+---------+
```

Figura 6 - Cabeçalho da tabela

Após a construção da tabela acima, o passo seguinte é fazer o treinamento. A função fit é responsável por essa etapa, como mostra a Figura 7.

```
pipelineFit = pipeline.fit(data_sample)
```

Figura 7 - Treinamento

Por fim, a Figura 8, mostra o código para fazer a predição.

```
dataset = pipelineFit.transform(data_sample)
```

Figura 8 -

Com o *dataset* feito, para que houvesse um código para teste e outro para treino os dados foram separados aleatoriamente numa proporção de 80% dos dados para treinamento e 20% para testar o treino. A Figura 9 mostra como foi feita essa repartição.

```
trainingData, testData = dataset.randomSplit([0.8, 0.2], seed = 100)8, 0.2], seed=100)
```

Figura 9 - Separação entre os dados de treinamento e teste

O treinamento dos dados é feita de duas formas distintas. A primeira é utilizando a regressão logística. A Figura 10 exemplifica o código desse treinamento e predição.

```
lr = LogisticRegression(maxIter=20, regParam=0.3, elasticNetParam=0)
lrModel = lr.fit(trainingData)
predictions = lrModel.transform(testData)
```

Figura 10 - Modelo de regressão logística

A segunda forma utilizada para fazer o treino e a predição foi utilizar o *Naive Bayes*. O código na Figura 11 representa essa situação.

```
nb = NaiveBayes(smoothing=1)
model = nb.fit(trainingData)
predictions = model.transform(testData)
```

Figura 11 - Modelo *Naive Bayes*

2.3 Teste de acurácia

Para verificar qual a acurácia dos modelos acima (regressão logística e *Naive Bayes*) o código da Figura 12 foi executado.

```
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)
```

Figura 12 - Verificação da acurácia

2.4 Teste com texto próprio

A última etapa foi testar o treinamento com um database de reviews feita posteriormente. Para tanto, utilizou-se o código da Figura 13. Primeiramente criou-se um banco de dados utilizando um gerador de *dataframe* do próprio *pyspark*. Com os dados em mão, próximo passo foi aplicar no treinamento feito anteriormente e assim analisar o resultado da predição de cada resenha.

```
teste = [
    Row(review_body='this product is garbage', star_rating=1),
    Row(review_body='this product is very good', star_rating=1)
]

df_teste = spark.createDataFrame(teste)

dataset_teste = pipelineFit.transform(df_teste)
predictions_teste = lrModel.transform(dataset_teste)
```

Figura 13 - Criação de *reviews* para teste e predição com o modelo de regressão logística

3 COMPARAÇÃO DE TESTE DE ACURÁCIA

Com o código da Figura 12, foi possível gerar a acurácia de ambos os modelos, regressão logística e Naive Bayes. O resultado pode ser visto na Tabela 1.

Modelo	Acurácia
Regressão Logística	0.8603493125232257
Naive Bayes	0.8451164791935583

Tabela 1 - Verificação da acurácia

Como é possível observar os valores da acurácia foram bastante semelhantes, sendo que a regressão logística se mostrou levemente melhor nesse caso.

4 RESULTADOS

O resultado obtido na predição das *reviews* de teste não tiveram um bom resultado, como mostra a Figura 14. É possível observar que o primeiro comentário é uma avaliação positiva e recebeu um valor positivo no "*star_rating*",

porém sua predição deu positiva (o "*prediction*" segue o mesmo valor do "*label*"). Porém, a segunda review que possui um comentário negativo recebeu uma "star_rating" positiva erroneamente, e sua predição não teve um resultado satisfatório, uma vez que retornou que a resenha era positiva. Essa situação mostra que a predição não está funcionando perfeitamente.

```
+-----------+------------------------+------------------------------+-----+----------+
|star_rating|             review_body|                   probability|label|prediction|
+-----------+------------------------+------------------------------+-----+----------+
|          1|this product is very good|[0.9169973494035765,0.08300...|  0.0|       0.0|
|          1|  this product is garbage|[0.8368225862499528,0.16317...|  0.0|       0.0|
+-----------+------------------------+------------------------------+-----+----------+
```

Figura 14 - Tabela com os *reviews* de teste com resultado da predição

5 CONCLUSÃO

As análise de avaliação das resenhas de produtos podem ser feitas de várias formas. A utilizada neste projeto busca decifrar a resenha como positiva ou negativa dependendo do texto redigido pelo cliente. O modelo de predição aqui usado não se mostrou eficiente em todos os casos de análise. Em situações onde a resenha é negativa, ocorreu maior chance de erro. Essa inconveniência pode ter diversas causas, um dos possíveis problemas vem do fato das resenhas positivas ter maior volume quando comparado às resenhas negativas, uma proporção de aproximadamente 88% positivo para 12% negativas, fazendo com que os resultados fiquem viciados. Outro fator que pode ter influenciado este problema é o fato de ter um banco de dados pequeno, ou seja, não há dados suficiente para um treinamento eficiente.

6 REFERÊNCIAS

[1] AWS. Amazon Customer Reviews Dataset. Disponível em: <https://registry.opendata.aws/amazon-reviews/>. Acesso em 27 de novembro de 2018.

[2] AWS. Amazon Customer Reviews Dataset (README). Disponível em <https://s3.amazonaws.com/amazon-reviews-pds/readme.html>. Acesso em 27 de novembro de 2018.

[3] LI, Susan. Multi-Class Text Classification with PySpark. Towards Data Science. Toronto Canada, 19 de fevereiro. Disponível em: <https://towardsdatascience.com/multi-class-text-classification-with-pyspark-7d78d022ed35>. Acesso em 27 de novembro de 2018.

[4] Spark. pyspark.ml package. Disponível em: <http://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html>. Acesso em 27 de novembro de 2018.