versao-serial-04-05

May 4, 2018

In [2]:
```
#importa pacotes para usar SQL no contexto do Spark
from pyspark.sql import SQLContext
from pyspark.sql.types import *
sqlContext = SQLContext(sc)
```

In [3]:
```
#roda e lê a base
data = sqlContext.read.load('file:///C:/Spark/projetos/Data/base.csv',
                            format='com.databricks.spark.csv',
                            header='true',
                            inferSchema='true')
```

In [4]:
```
#mostra as duas primeiras linhas da base
data.show(2)
```

```
+--------+-------------+--------+--------+------------------+-------+----------+----------------
|   CAMIS|          DBA|    BORO|BUILDING|            STREET|ZIPCODE|     PHONE|CUISINE DESCRIPT
+--------+-------------+--------+--------+------------------+-------+----------+----------------
|50074025|    LILA CAFE|BROOKLYN|     911|        DEKALB AVE|  11221|3475292886|          Caribb
|41573314|SABANA LOUNGE|   BRONX|    1460|WESTCHESTER AVENUE|  10472|7186207100|            Span
+--------+-------------+--------+--------+------------------+-------+----------+----------------
only showing top 2 rows
```

In [5]:
```
#imprime os rótulos e tipos de dado da base
data.printSchema()
```

```
root
 |-- CAMIS: integer (nullable = true)
 |-- DBA: string (nullable = true)
 |-- BORO: string (nullable = true)
 |-- BUILDING: string (nullable = true)
 |-- STREET: string (nullable = true)
 |-- ZIPCODE: string (nullable = true)
 |-- PHONE: string (nullable = true)
 |-- CUISINE DESCRIPTION: string (nullable = true)
 |-- INSPECTION DATE: string (nullable = true)
 |-- ACTION: string (nullable = true)
```

```
 |-- VIOLATION CODE: string (nullable = true)
 |-- VIOLATION DESCRIPTION: string (nullable = true)
 |-- CRITICAL FLAG: string (nullable = true)
 |-- SCORE: integer (nullable = true)
 |-- GRADE: string (nullable = true)
 |-- GRADE DATE: string (nullable = true)
 |-- RECORD DATE: string (nullable = true)
 |-- INSPECTION TYPE: string (nullable = true)
```

```
In [6]: #exclui os dados não rotulados
        data = data[~data['CRITICAL FLAG'].isin(['Not Applicable'])]
        #retira as colunas da tabela que não serão utilizadas na classificação
        drop_list = ['CAMIS', 'DBA', 'BORO', 'BUILDING', 'STREET', 'ZIPCODE','CUISINE DESCRIPTIO
```

```
In [7]: #mostra as colunas que serão utilizadas na classificação
        #CRITICAL FLAG É A COLUNA COM OS CLASSIFICADORES
        data = data.select([column for column in data.columns if column not in drop_list])
        data.show(5)
```

```
+--------------------+-------------+
|VIOLATION DESCRIPTION|CRITICAL FLAG|
+--------------------+-------------+
| Non-food contact ...| Not Critical|
| Non-food contact ...| Not Critical|
| Non-food contact ...| Not Critical|
| Non-food contact ...| Not Critical|
| Food contact surf...|     Critical|
+--------------------+-------------+
only showing top 5 rows
```

```
In [8]: #imprime os rótulos e tipos de dado da base
        data.printSchema()
```

```
root
 |-- VIOLATION DESCRIPTION: string (nullable = true)
 |-- CRITICAL FLAG: string (nullable = true)
```

```
In [9]: #conta quantas linhas há para cada classificador
        from pyspark.sql.functions import col
        data.groupBy("CRITICAL FLAG") \
            .count() \
            .orderBy(col("count").desc()) \
            .show()
```

```
+-------------+------+
|CRITICAL FLAG| count|
+-------------+------+
|     Critical|204546|
| Not Critical|162077|
+-------------+------+
```

```
In [10]: #conta as principais descrições
         data.groupBy("VIOLATION DESCRIPTION") \
             .count() \
             .orderBy(col("count").desc()) \
             .show(5)
```

```
+--------------------+-----+
|VIOLATION DESCRIPTION|count|
+--------------------+-----+
| Non-food contact ...|52814|
| Facility not verm...|38420|
| Evidence of mice ...|26858|
| Food not protecte...|25364|
| Food contact surf...|25240|
+--------------------+-----+
only showing top 5 rows
```

```
In [11]: #importa pacotes para tokenizar, remover stopwords e vetorizar
         from pyspark.ml.feature import RegexTokenizer, StopWordsRemover, CountVectorizer
         from pyspark.ml.classification import LogisticRegression
         # regular expression tokenizer
         regexTokenizer = RegexTokenizer(inputCol="VIOLATION DESCRIPTION", outputCol="words", pa
         # stop words
         #usando stop list of 25 semantically non-selective words which are common in Reuters-RC
         add_stopwords = ["a","an", "and", "are", "as", "at", "be", "by", "for", "from", "has",
         stopwordsRemover = StopWordsRemover(inputCol="words", outputCol="filtered").setStopWord
         # bag of words
         countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=100
```

```
In [12]: #cria rótulos númericos para os classificadores
         from pyspark.ml import Pipeline
         from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
         label_stringIdx = StringIndexer(inputCol = "CRITICAL FLAG", outputCol = "label")
         pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, countVectors, label_strin
```

```
In [13]: # Forma o pipeline to para treinar os documentos.
         pipelineFit = pipeline.fit(data)
         dataset = pipelineFit.transform(data)
         dataset.show(5)
```

```
+--------------------+------------+--------------------+--------------------+----------------
|VIOLATION DESCRIPTION|CRITICAL FLAG|               words|            filtered|            featu
+--------------------+------------+--------------------+--------------------+----------------
| Non-food contact ...| Not Critical|[non, food, conta...|[non, food, conta...|(528,[0,1,2,4,5,6
| Non-food contact ...| Not Critical|[non, food, conta...|[non, food, conta...|(528,[0,1,2,4,5,6
| Non-food contact ...| Not Critical|[non, food, conta...|[non, food, conta...|(528,[0,1,2,4,5,6
| Non-food contact ...| Not Critical|[non, food, conta...|[non, food, conta...|(528,[0,1,2,4,5,6
| Food contact surf...|     Critical|[food, contact, s...|[food, contact, s...|(528,[1,2,5,6,7,3
+--------------------+------------+--------------------+--------------------+----------------
only showing top 5 rows
```

In [14]:
```
# Define as sementes para reprodutibilidade
(trainingData, testData) = dataset.randomSplit([0.7, 0.3], seed = 100)
print("Training Dataset Count: " + str(trainingData.count()))
print("Test Dataset Count: " + str(testData.count()))
```

```
Training Dataset Count: 257050
Test Dataset Count: 109573
```

In [16]:
```
#calcula o tf-idf
from pyspark.ml.feature import HashingTF, IDF
hashingTF = HashingTF(inputCol="filtered", outputCol="rawTF", numFeatures=10000)
tf = hashingTF.transform(trainingData)
idf = IDF(inputCol="rawTF", outputCol="IDF", minDocFreq=5) #minDocFreq: remove termos s
idfModel = idf.fit(tf)
tfidf = idfModel.transform(tf)
pipeline = Pipeline(stages=[regexTokenizer, stopwordsRemover, hashingTF, idf, label_str
```

In [17]:
```
#Meu modelo fará previsões e pontuação no conjunto de testes
#Mostra as 30 principais previsões da maior probabilidade.

lr = LogisticRegression(maxIter=20, regParam=0.3, elasticNetParam=0)
lrModel = lr.fit(trainingData)
predictions = lrModel.transform(testData)
predictions.filter(predictions['prediction'] == 0) \
    .select("VIOLATION DESCRIPTION","CRITICAL FLAG","probability","label","prediction")
    .orderBy("probability", ascending=False) \
    .show(n = 30, truncate = 30)
```

```
+-----------------------------+------------+------------------------------+-----+----------+
|         VIOLATION DESCRIPTION|CRITICAL FLAG|                   probability|label|prediction|
+-----------------------------+------------+------------------------------+-----+----------+
|Filth flies or food/refuse/...|     Critical|[0.9660886616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.9660886616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.9660886616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.9660886616772627,0.03391...|  0.0|       0.0|
```

```
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
|Filth flies or food/refuse/...|     Critical|[0.966088616772627,0.03391...|  0.0|       0.0|
+------------------------------+-------------+-----------------------------+-----+----------+
only showing top 30 rows
```

In [19]:
```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")
evaluator.evaluate(predictions)
```

Out[19]: 1.0

In [20]:
```python
#soma os vetores
from pyspark.ml.linalg import SparseVector, DenseVector

# Representacao do vetor de classes
df = dataset.select('label','features')
vetor_de_classes = df.rdd.mapValues(lambda v: v.toArray()) \
    .reduceByKey(lambda x, y: x + y) \
    .mapValues(lambda x: DenseVector(x)) \
    .toDF(["label", "features_sum"])

vetor_de_classes.show(2)
```

```
+-----+--------------------+
|label|        features_sum|
+-----+--------------------+
|  0.0|[213413.0,297513...|
|  1.0|[356079.0,119209...|
+-----+--------------------+
```

In [21]:
```
# preparo para o calculo da distancia euclidiana para classe 0.0
array0 = vetor_de_classes.filter('label = 0.0').collect()[0]['features_sum']
print(array0)
```

```
[213413.0,297513.0,121387.0,220272.0,55598.0,25257.0,28358.0,25570.0,8605.0,66065.0,0.0,62059.0,
```

In [22]:
```
# preparo para o calculo da distancia euclidiana para classe 1.0
array1 = vetor_de_classes.filter('label = 1.0').collect()[0]['features_sum']
print(array1)
```

```
[356079.0,119209.0,230316.0,0.0,105628.0,123876.0,113290.0,110890.0,111028.0,52814.0,115260.0,43
```

In [23]:
```
# calcula a distancia euclidiana
from scipy.spatial import distance
from pyspark.sql.types import FloatType
from pyspark.sql.functions import udf

euc0 = udf(lambda features: distance.euclidean(features, array0), FloatType())
euc1 = udf(lambda features: distance.euclidean(features, array1), FloatType())

df = df.withColumn('distance0', euc0(df.features))
df = df.withColumn('distance1', euc1(df.features))
df.show(5)
```

```
+-----+--------------------+---------+---------+
|label|            features|distance0|distance1|
+-----+--------------------+---------+---------+
|  1.0|(528,[0,1,2,4,5,6...| 530766.2| 596561.5|
|  1.0|(528,[0,1,2,4,5,6...| 530766.2| 596561.5|
|  1.0|(528,[0,1,2,4,5,6...| 530766.2| 596561.5|
|  1.0|(528,[0,1,2,4,5,6...| 530766.2| 596561.5|
|  0.0|(528,[1,2,5,6,7,3...| 530767.7|596566.25|
+-----+--------------------+---------+---------+
only showing top 5 rows
```