Basic data preparation in Pyspark — Capping, Normalizing and Scaling

Soumya Ghosh Follow
Mar 21, 2018 · 3 min read

In this blog, I'll share some basic data preparation stuff I find myself doing quite often and I'm sure you do too. I'll use Pyspark and I'll cover stuff like removing outliers and making your distributions normal before you feed your data into any model, be it linear regression or nearest neighbour searches.

You don't always need to remove outliers and skewness from your data. It highly depends on how you're going to use it. For example, algorithms like decision trees arnt affected by outliers, but algorithms like linear regression or even neural nets expect your data to have somewhat normal distributions. Scaling also has a big effect on any mode which calculates distances between observations. I have noticed pretty distinct jumps in model performance before and after removing skewness from my data in some projects. Lets start.

```
# importing some libraries

import numpy as np
from pyspark.sql import functions as F
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.feature import VectorAssembler

# checking if spark context is already created

print(sc.version)

# reading your data as a dataframe

df = sqlContext.read.format("csv") \
   .options(header='true', inferschema='true') \
   .load(os.path.realpath("your_csv.csv"))
```

I'd suggest to investigate each and every column individually, but for simplicity let me assume that all my features have highly skewed distributions and long tails on both ends.

So I've decided to cap all my columns at 1st and 99th percentile, that is I'll replace any value below the first percentile with the 1st percentile and all values above the 99th percentile that the 99th percentile. Instead of having a solution this generic, it is better to have feature-wise cutoffs.

After that I'll do a log transformation — log(x+1) on each feature, the one is there so that I dont run into an error when I do log(0). You would probably want to do a log transformation if the distribution is right skewed and an exponential transformation if left skewed. Other transformations like that boxcox is also pretty popular too.

First I'll calculate the 1st and 99th percentile for every feature and strore them in the dictionary d.

```
# empty dictionary d

d = {}

# Fill in the entries one by one

for col in df.columns[1:-3]:
      d[col] = df.approxQuantile(col,[0.01,0.99],0.25)
      print(col+" done")
```

Im using approxQuantile, otherwise it just takes too much time to calculate the percentiles if you have a huge dataset.

```
# looping through the columns, doing log(x+1) transformations

for col in df.columns:
    df_new = df.withColumn(col, \
    F.log(F.when(df[col] < d[col][0],d[col][0])\
    .when(df[col] > d[col][1], d[col][1])\
    .otherwise(df[col] ) +1).alias(col))
    print(col+" done)
```

You can use the approxQuantile function again to check if the percentiles are more evenly distributed now. Now that Im done with the log transformation, Ill move onto scaling the data — making sure all the features are between 0 and 1. For that I'll use the VectorAssembler(), it nicely arranges your data in the form of Vectors, dense or sparse before you feed it to the MinMaxScaler() which will scale your data between 0 and 1.

```
assembler = VectorAssembler().setInputCols\
            (df_new.columns).setOutputCol("features")
transformed = assembler.transform(df_new)
scaler = MinMaxScaler(inputCol="features",\
         outputCol="scaledFeatures")
scalerModel =  scaler.fit(transformed.select("features"))
scaledData = scalerModel.transform(transformed)
```

I'm almost done. Now you'll notice that what these functions have done is transformed your data and joined it with your original dataframe df_new. The final features are now in the form of a list in the "features" column. All thats left is make a dataframe out of them.

```
def extract(row):
    return (row.pmid, )+tuple(row.scaledFeatures.toArray().tolist())

final_data = scaledData.select("pmid","scaledFeatures").rdd\
               .map(extract).toDF(df.columns)
```

Finally I'll save the data as a csv. Notice that Im repartitioning the data so that I get one file instead of a lot of part files.

```
# saving the file

final_data.repartition(1).write.csv("file_name.csv")
```

And we're done. If this was helpful to you, please leave a like and a comment.

Python Spark Data Analysis Pyspark Data Preparation

About Help Legal