BDSE Part 2

Quirijn Langedijk

500782499

quirijn.langedijk@hva.nl

email@quirijnlangedijk.nl

Inhoud

**Geen inhoudsopgavegegevens gevonden.**

# Summary

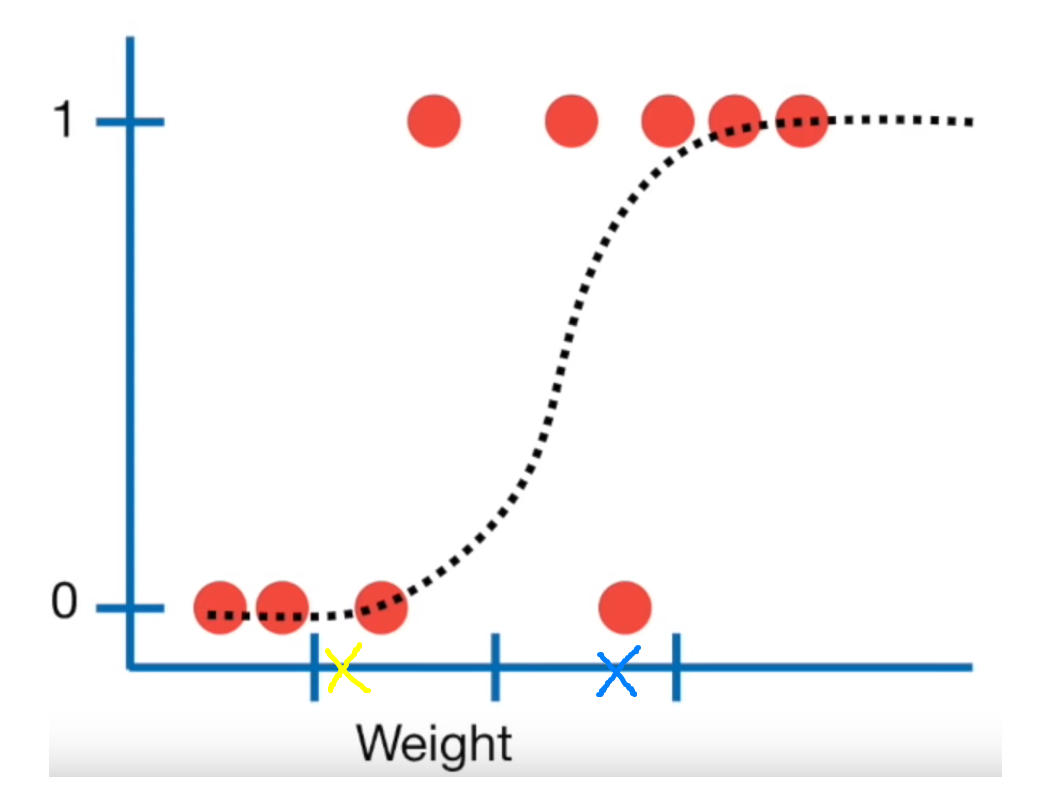
# Introduction

With this assignment, it was our task to do a couple of things; Create a visualization of a dataset containing around 600.000 hotel reviews, use Spark to create a machine learning algorithm to classify the reviews as positive or negative, and create a simulation on how to solve storage problems.

# Background

For this assignment, I used Spark to implement a Logistic Regression model. Where Linear Regression is able to predict a value based on an input value (for example size based on weight, infinite Y values), Logistic Regression can only predict if a value is True or False (Y between 0 and 1).

Instead of having a straight line, Logistic Regression uses a Sigmoid function to calculate probability, which is displayed as the following:



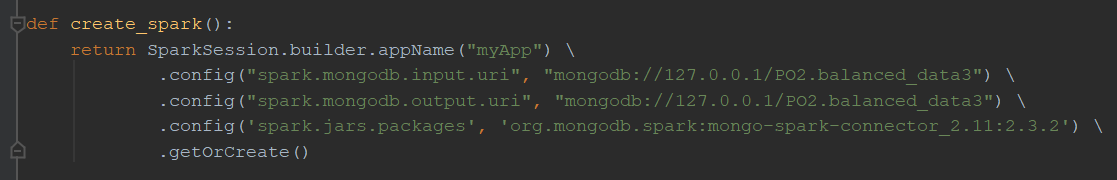
With this graph, we can predict the probability someone is obese by their weight. In the example above, if 0 is classified as not obese, and 1 as obese, there’s a big chance the blue cross is classified as obese, while the chance is lower with the yellow cross.

Contains theory about the models

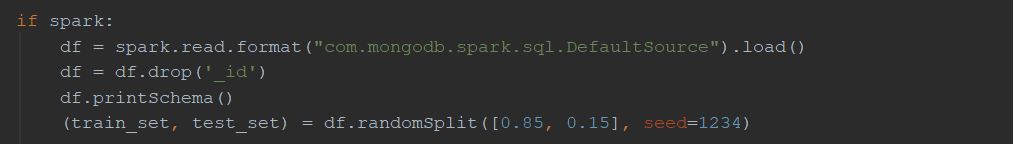
# Methods

## Spark Logistic Regression

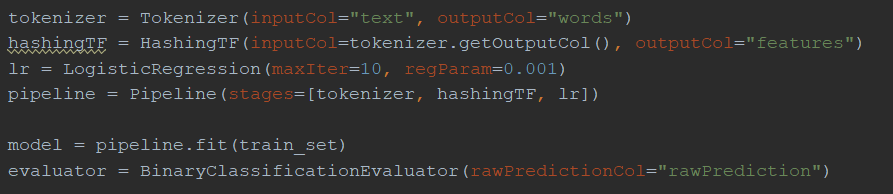
To classify the reviews as positive or negative, I used a Spark Pipeline with Logistic Regression. We first create a SparkSession to retrieve data from the MongoDB:



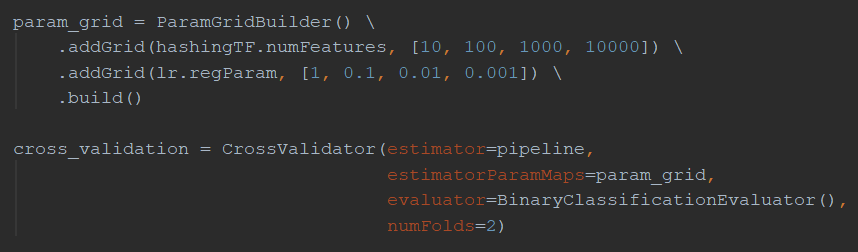
Then load the data, drop the \_id column (since it’s useless for classification), and divide the SparkDF into a train and test set:



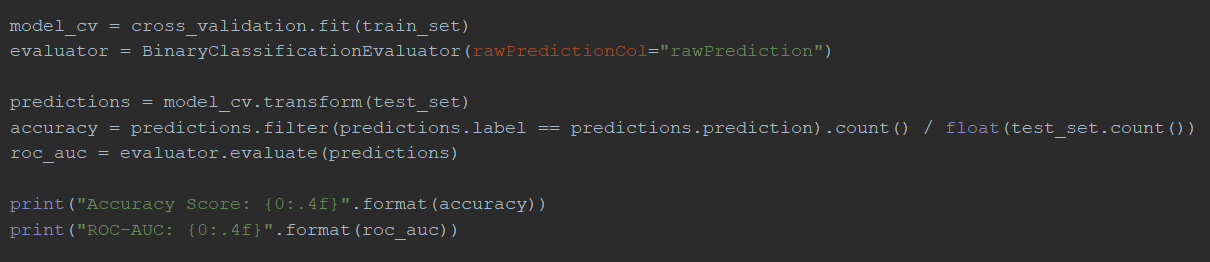
Were then ready to build the Pipeline, since were processing text, we will have to create values for each word before we can use them in Logistic Regression. This is done by tokenizing and hashing the words.



To optimize our model, we use cross validation, this means Spark will try out different combinations of the given parameters, and picks out the ones that fit the data the best:



Were then ready to fit the data, and test the model afterwards:

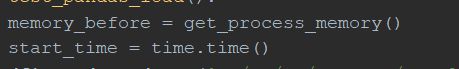


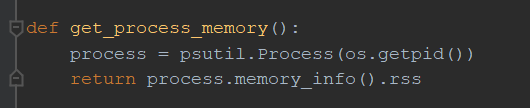
In the case of these parameters the Accuracy was 92.4%, ROC-AUC 96.7%, and train time 39 seconds.

## Spark …

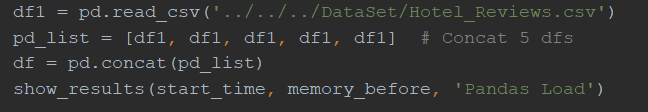
## Dask vs Pandas

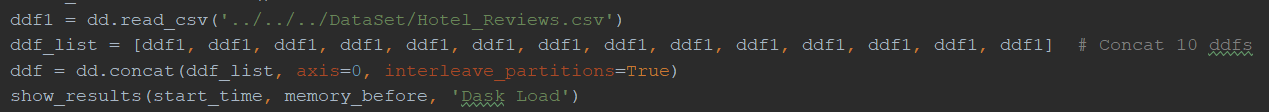
To test the RAM usage of Pandas and Dask, I first took the time and current memory usage:



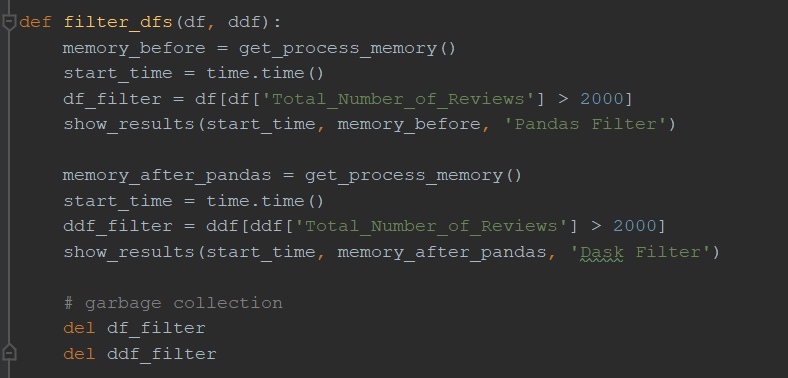


Then I read the dataframe and Dask dataframe, and concat them 5/10 times to increase the size of the (Dask) dataframe:

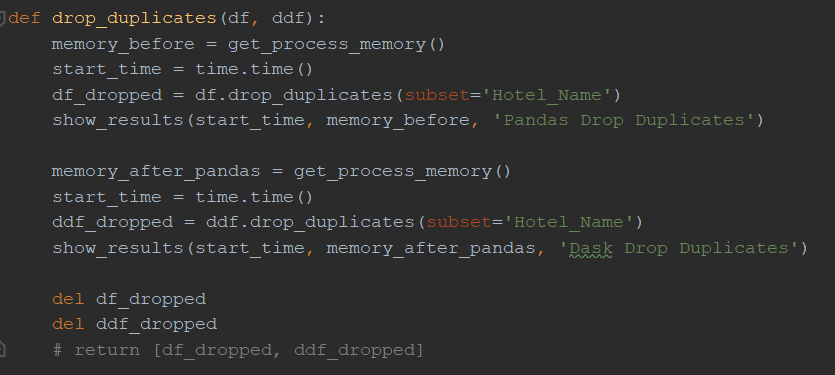


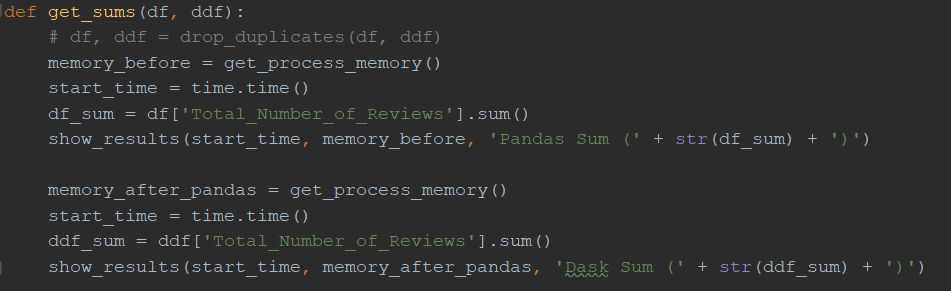


After that, I measured the time/RAM usage to filter the dataframes:

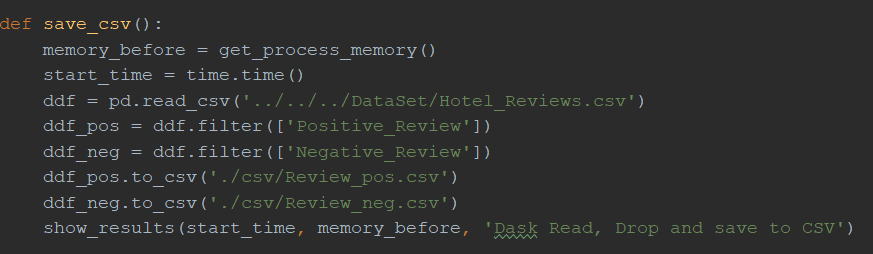


As well as drop the duplicates and get the sums:





I then saved the positive/negative rows:



## Visualisation

Can contain multiple subsections

Screenshots of code, only when relevant

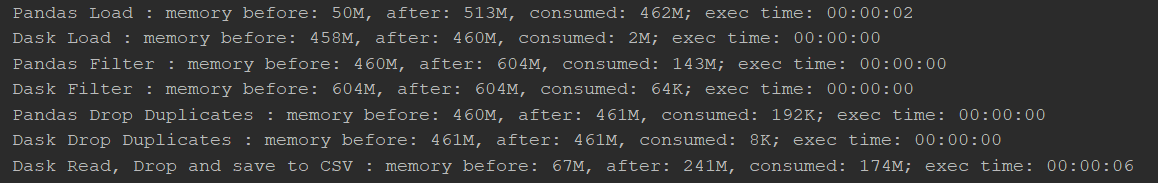
# Results

## Spark Logistic Regression

|  |  |  |  |
| --- | --- | --- | --- |
| Number of rows | Accuracy Score | ROC-AUC | Train time |
| 1000 | 87.90% | 93.45% | 00:00:57 |
| 5000 | 90.58% | 95.55% | 00:01:04 |
| 10000 | 91.84% | 96.24% | 00:01:00 |
| 50000 | 93.57% | 98.05% | 00:01:28 |
| 100000 | 94.08% | 98.07% | 00:01:54 |

## Spark …

## Dask vs Pandas



As you can see, the ram usage is way lower with Dask than Pandas, Dask had no problem dealing with dataframes of 7+ million rows, whereas Pandas already maxed out at half of that.

## Visualisation

…

Contain relevant plots

# Conclusion

# Reference list