Predicting Cancellation of Hotel Reservation

A hotel chain is interested in using its records of reservations to predict the likelihood of a given reservation to be cancelled. It is helpful to the hotel to be prepared for that since it enables them to overbook and thereby fill the available space, or alternatively to decline a reservation to avoid the vacancy, or perhaps to take a non-refundable deposit in case of high likelihood of cancellation in busy times.

The dataset [[Hotel booking demand (kaggle.com)](https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand)] is too large for local analysis so it is resident in the Hadoop file system. The file I was using was a csv file with ~120000 records. Although this file might be appropriate for a local system, it contained records of only three years, so a distributed file system is more suitable.

To do this analysis I used the Pyspark ML library of Apache Spark. First the extensive preprocessing, and feature selection, and since this is a supervised classifier task since we do have cancellation data, I could have chosen any of the classifiers, so I started with the logistic regression.

Before working with the ml library, I needed to install the numpy package, which was not part of the default installation- on the master and both worker nodes. This had to be done each time I shut down the docker containers.

I did not find it necessary to use Spark sql, since any data wrangling could be done with the pyspark functions and methods.

As regards the feature selection, classifiers do not do well with features that have too many values, or values that are too rare, so after examining the data, (which is somewhat cumbersome in Pyspark) I chose ten fields as the relevant features: hotel type, month, repeat guest, number of prior cancellations, number of prior bookings, country, number of nights- weekday, number of nights- weekend, customer type, lead time. All of these are imported to Spark as string variables, and all of them would need to be converted to numericals: whether binary, or ordinal.

First check for Null values

dfnulls={col:df.filter(df[col].isNull()).count() for col in df.columns}

There were none in this dataset. To convert string variables of numbers into integer variables, there are several syntaxes, so I used, for example,

df=df.withColumn('is\_canceled', df.is\_canceled.cast('int'))

Because the lead time refers to number of days before check-in that the customer made the reservation, it is a number between 0 and several hundred. So I converted that variable to an numerical categorical at what I thought were meaningful split points:

buck=Bucketizer(splits=[0,7,14,50,100,365,float('Inf')], inputCol='lead\_time', outputCol='lead\_bins')

df=buck.setHandleInvalid('keep').transform(df)

df=df.drop('lead\_time')

To encode the remaining string variables into categorical numericals, I fit the StringIndexer function to the variables individually, such as:

hotelindexer=StringIndexer(inputCol='hotel', outputCol='hotelind')

hotelindfit=hotelindexer.fit(df)

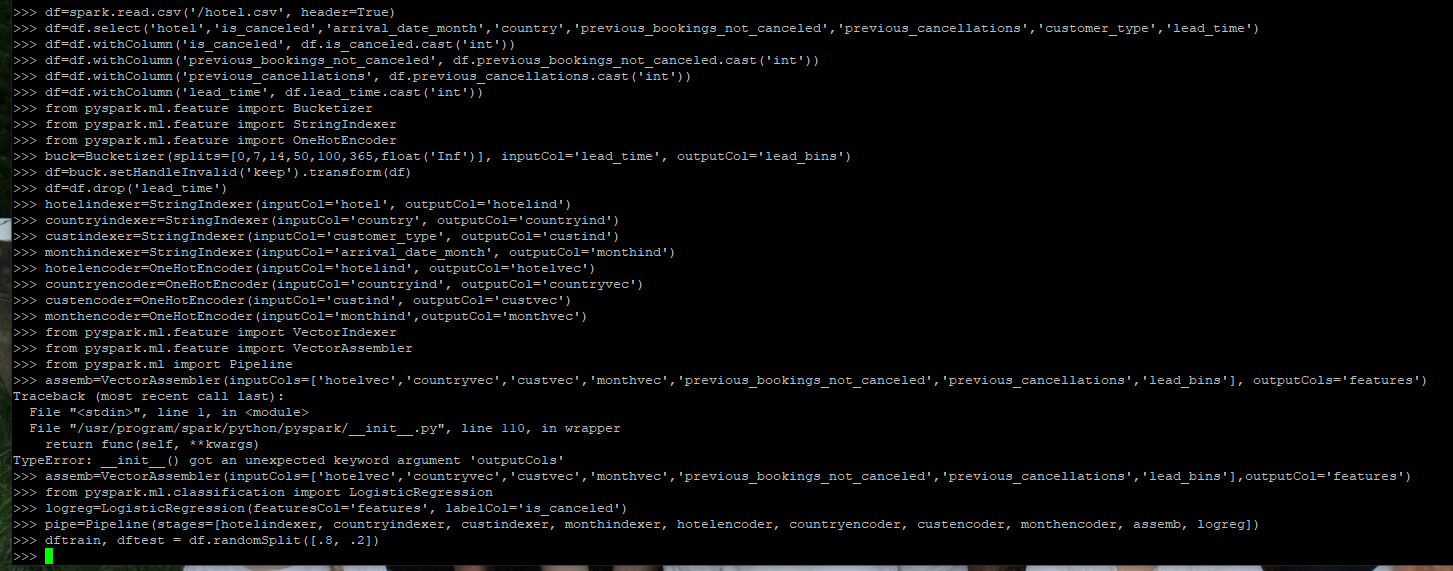
Once all the variables are encoded as categorical numericals, I use the One Hot Encoder to convert the whole thing to dummy variables, then through an Assembler to convert all of the dataframe features to a single vector. All of these functions are available in the pyspark ml library, once numpy is installed.

After all the preprocessing, we can run it through the classifier- or fit the Logistic Regression model to the dataset. However, before actually running any of the ml feature functions on the dataset, we need to split the full dataset into training and testing sets:

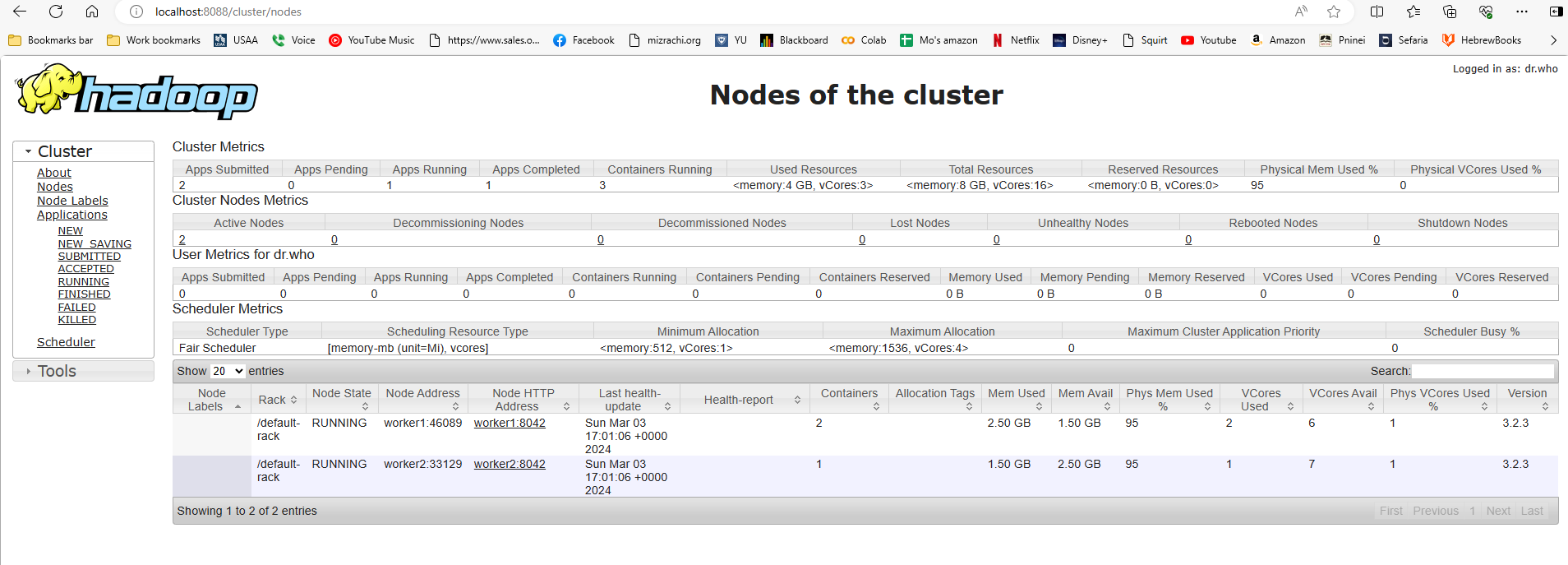
dftrain, dftest = df.randomSplit([.8, .2]) (I personally prefer 80/20 splits)

Happily, the pyspark Logistic Regression function does not require splitting off the target variable, is\_canceled, from the feature set- just to identify which column is the target (the label).

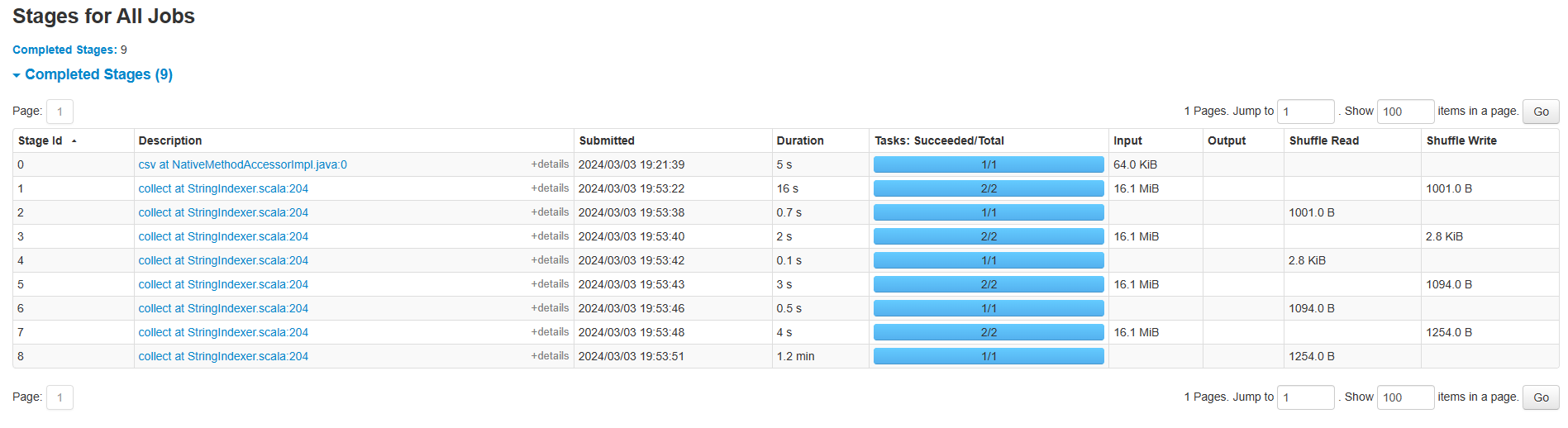
Since there are many steps to the preprocessing and we need to do them to the training set and testing set separately, we build the pipeline of feature encodings as described, and train the model on the training data, then produce the predicted target values from the testing data. Finally, using Pyspark’s Binary Classifier Evaluator we can check how good our model is. See screenshot of commands from one version of the task up until fitting the pipeline:



This was an exciting part of the project, anticipating the power of Spark’s bigdata processing capabilities to run through a job like this in no time- well, it took a while, cycled through several stages, and got up to stage 8 before crashing (freezing)- several times. So, I have nothing to report as an outcome. However I am confident that this is a limitation on the computing power of the VM configuration, or possibly my own computer, and that in principle, this method would work very well. I did wonder at the available storage and cloud-based resources, so I checked the Hadoop UI:



It isn’t so clear to me which part of this I should be monitoring, but the Memory Used, and Phys Mem Used 95% made me concerned. However, Spark shows no problems:



Which means that I couldn’t understand the point of trouble for the freezing.

Ignoring the trouble of the system crashing, the results of the analysis could be helpful, especially if we could dig into the classifier formula. Meaning- the pattern of hotel cancellations as related to some combination of factors, might be useful for hotel management to discover. Moving forward, several variables were excluded from this analysis, such as the booking method, room type, amount of deposit, and many others, might add to the quality of the model. Likewise, Pyspark has other classifiers, such as random forest, naïve bayes, KNN, k-means- all relatively easy to implement, and to hypertune.

It remains unclear to me how speedy the Spark system is- each step of processing took time. Referencing the Spark screenshot above, some steps in the model building took several seconds- to over a minute! Is this the actual pace of the processing? I have to suppose that for major, enterprise systems, the processing is much more rapid.

Personal reflections:

Each step in this assignment was like slogging through two feet of snow- how to get the dataset into the cloud- then where is it, and how to get it into the containers, and how to work with the variables, how to examine them and how to get the ml library to load, and each time it crashed necessitated doing much of it all over again. Still, every minor success was extremely gratifying, since this entire world of distributed computing was absolutely blank to me before this course, and I was very aware of how crucial this is to my involvement in the field. I, along with all of your students, am very appreciative of your helpfulness, responsiveness, and desire to see us achieve and succeed (even if some of us didn’t really), constructive feedback, and how much you gave of yourself to support us. Thank you for that and everything else.