MLlib PySpark Algorithm on DataProc of Google Cloud Platform

Having completed my data science internship working on AWS environment, I was intrigued by the cloud services and its disrupting capabilities on Machine Learning. While the existing counterparts are pretty much scalable and widely used, Google Cloud Platform is the newest kid in the block and has increasingly become the first choice for cloud services and ML applications on cloud.

Introduction

This project is an implementation of PySpark' s MLlib application over GCP's DataProc Platform. It briefly illustrates ML cycle from creating clusters to deploying the ML algorithm. The dataset used is a healthcare dataset as describe in the picture, where we aim to predict if the person had a stroke or not.

```
In [17]: pd.DataFrame(df.take(5), columns=df.columns)
```

Out[17]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	None	0
1	30468	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	16523	Female	8.0	0	0	No	Private	Urban	110.89	17.6	None	0
3	56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	46136	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	None	0

There are null values for BMI, smoking status and gender which are treated separately mentioned later.

Necessary encoding and mapping of categorical values are done, mentioned as well later.

Google Cloud Platform - DataProc

GCP has various tools and compute engines like Dataflow, AutoML etc. that one can leverage from depending upon their project requirement and complexities. For this project, I have used DataProc. DataProc is a fast-scalable cloud service to run Apache Spark and Hadoop through GCP. With built-in integration with Cloud Storage, it allows autoscaling, resizing clusters, developer tools and several optional components (open source) related to Hadoop and Apache Spark ecosystem.

The steps to set up environment in GCP would be majorly the 3 below:

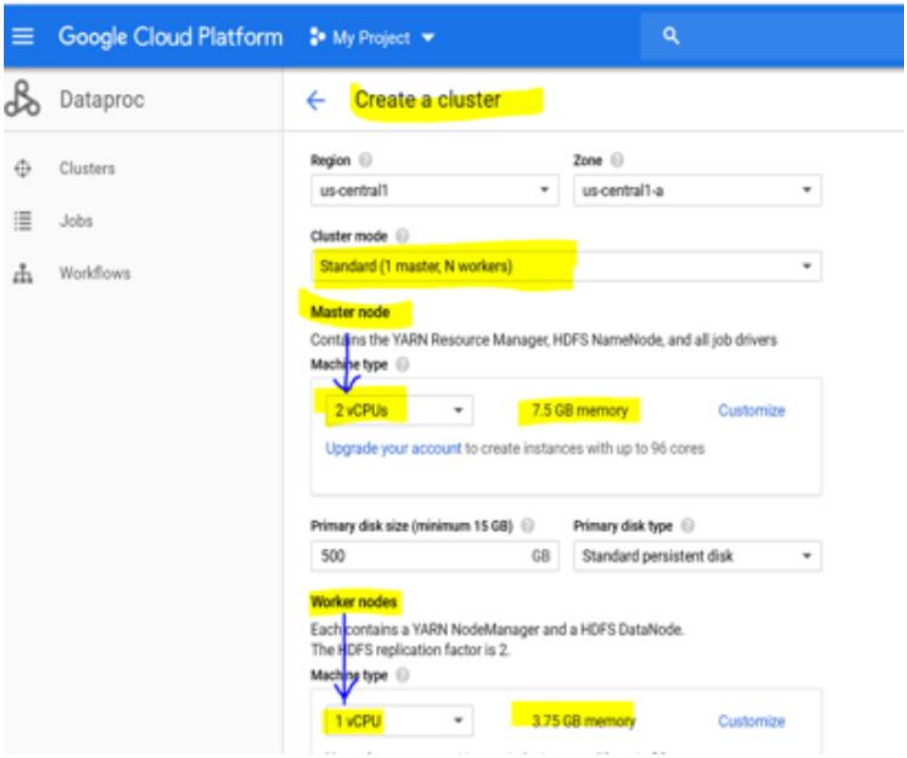
1) Create VM clusters with VM instances

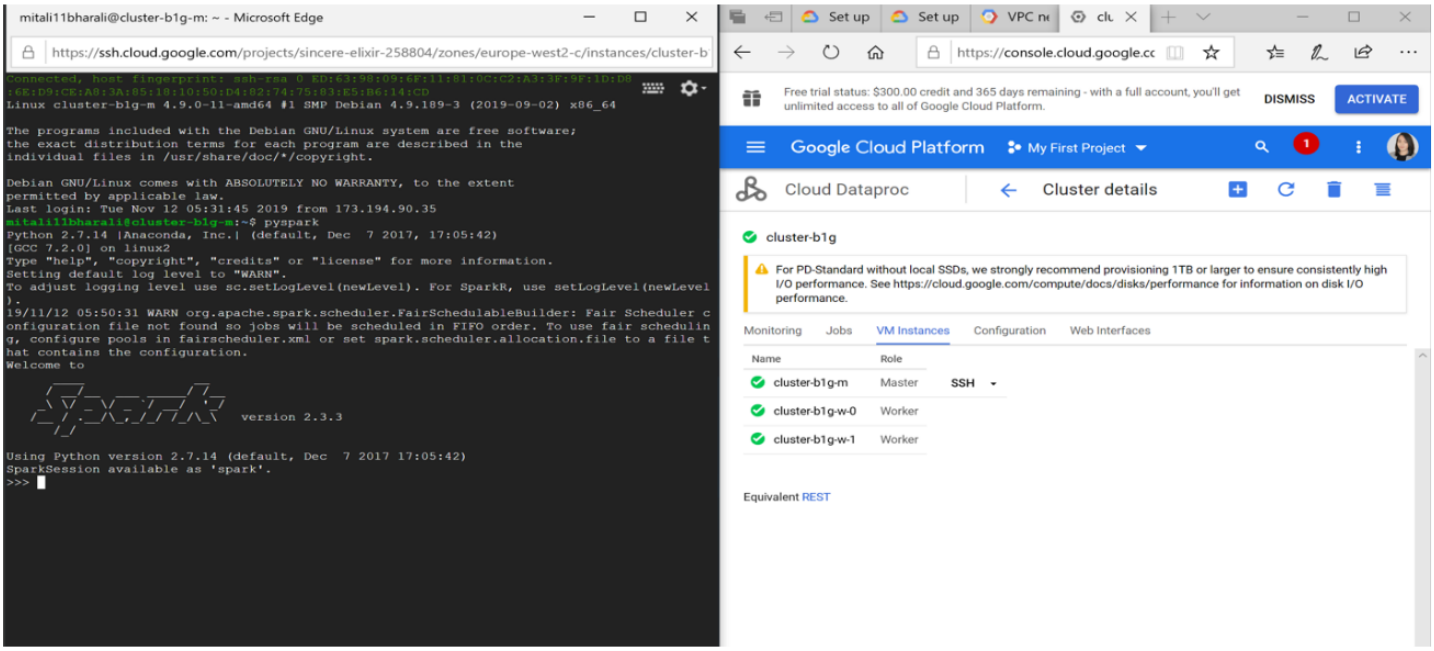
2) Create Firewall Rule

3) Set up External IP Address

VM Instance in a Cluster

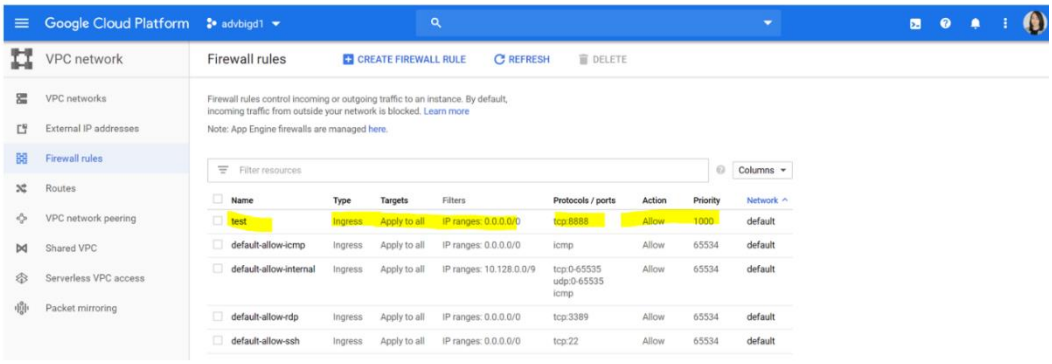
For this project I build one master node that requests the recourses for the cluster to make them available for the spark driver and 2 worker nodes. Specification of which can be chosen as per requirement, mine as described in the picture below. Instantiate the SSH of the worker node and run Pyspark to set the version to 3.6 of python and spark version more than 2.2.

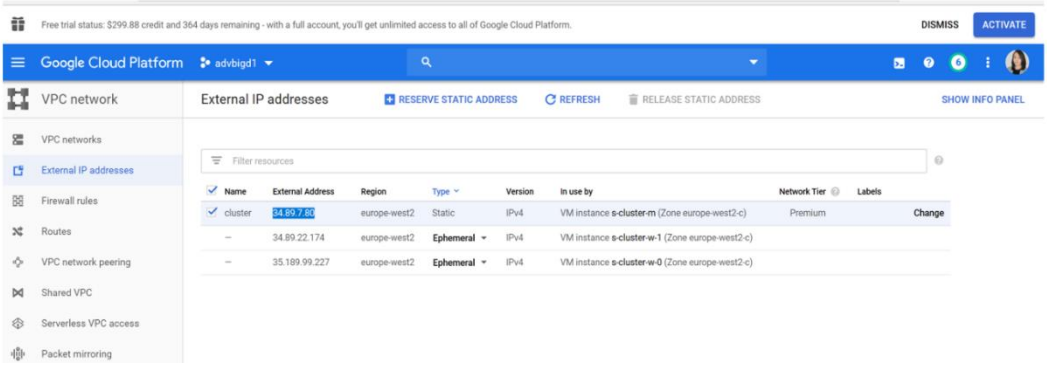




Firewall Rule and External IP Address

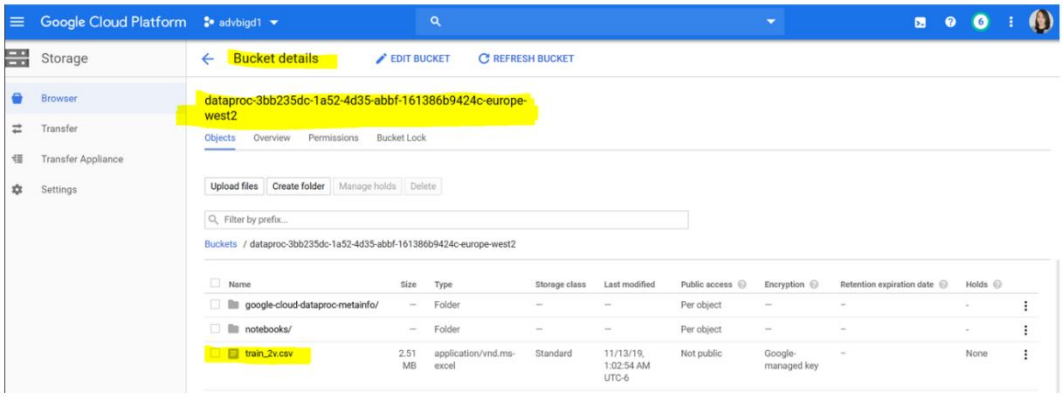
Over the VM instance, I set the Firewall rule as test, with port as 8888 and source IP address as 0.0.0.0, which is later used to instantiate the jupyter notebook on the web. Later, from the External IP address tab, note the IP address of the master node, which is used to open the jupyter notebook on web.





Bucket

Just like as in AWS, we must use the storage tab to create a bucket and upload necessary files to be used in the project. This is where you can also define pricing rules where you can allocate pricing for every download of the files or let the clients who use the file pay for the project rather than the developer.



Once done, you can open the Jupyter file from the web browser and start off with the project.

Pyspark in Jupyter

Following packages were used in my project

I. Import Packages

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import re
```

In [2]:
```
from pyspark.sql import SQLContext
from pyspark.sql import DataFrameNaFunctions
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import Binarizer
from pyspark.ml.feature import OneHotEncoder, VectorAssembler, StringIndexer, VectorIndexer
from pyspark.ml.classification import RandomForestClassifier
from pyspark.sql.functions import avg
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
from sklearn.model_selection import train_test_split
from collections import Counter
from sklearn import preprocessing
```

Instantiate a Spark Session

II. Spark Session

In [3]:
```
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
#sc = SparkContext('local')
#spark = SparkSession(sc)
```

In [4]:
```
spark = SparkSession.builder.appName("Predict Stroke Data").getOrCreate()
```

In [5]:
```
input_dir = 'gs://data_stroke_1/'
df = spark.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load(input_dir+'train_2v.csv')
```

Here, we can read the file that we saved in the bucket of our GCP storage using databricks.spark.

Once done, I converted the spark dataframe into Pandas to support all operations.

Data Preprocessing

The two major steps here is the *Imputation* and *Sampling*.

Imputation with KNN and Multinomial Regression (without imputer library)

This is the key area of the project where I implemented functions and logic of my own.

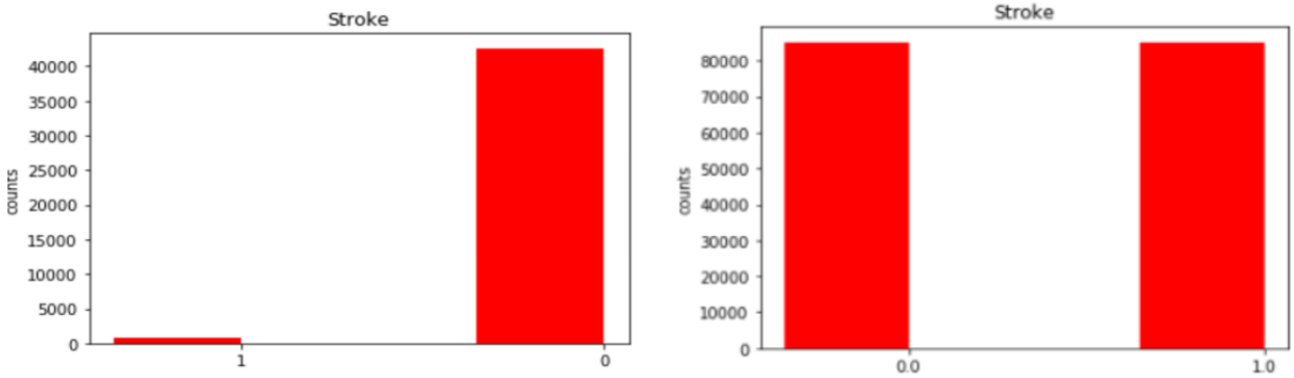
I used ML supervised learning algorithms without library imputer function. I made a loop function to iterate over every row with NaN values and replace/impute those values with the nearest neighbor value (knn= 3).

Similarly, for a categorical value, I used Label Encoder to convert into numeric and use a Logistic multinomial regression to impute the NaN values.

Sampling the Imbalances

While there are several ways to perform sampling to overshadow the imbalances of a dataset, most effective once for me is still weighted cost function in the regression.

However, for this project I wanted to implement and illustrate Smote Oversampling to sample and resampled a highly imbalanced dataset, given our target variable is itself extremely imbalanced.



Before Sampling

After Sampling

(0 – no stroke; 1 – stroke)

Building Spark ML Pipeline

I used Binarizer to build an assembler, which assembles all column together to predict the target variable and produce one vector as feature.

```
featureColumns = ['id', 'age', 'hypertension', 'heart_disease', 'ever_married', 'Residence_type', 'avg_glucose_level', 'bmi', 'ge
```

```
binarizer = Binarizer(threshold=0.0, inputCol="stroke", outputCol="label")
binarizedDF = binarizer.transform(imputeDF_f)
binarizedDF = binarizedDF.drop('stroke')
```

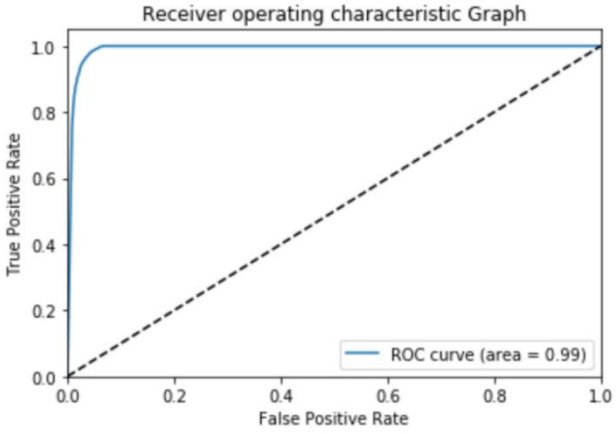
```
assembler = VectorAssembler(inputCols = featureColumns, outputCol = "features")
assembled = assembler.transform(binarizedDF)
print(assembled)
```

```
DataFrame[Residence_type: double, age: double, avg_glucose_level: double, bmi: double, ever_married: double, gender_imputed: do
uble, heart_disease: double, hypertension: double, id: double, smoke_formerly smoked: double, smoke_never smoked: double, smoke
_smokes: double, work_Govt_job: double, work_Never_worked: double, work_Private: double, work_Self-employed: double, work_child
ren: double, label: double, features: vector]
```

Modelling

I kept the modelling pretty much simple since, imputation and sampling is supposed to cancel out any anomalies of the dataset and I want to capture the true information of the treated dataset as is.

Using simple decision tree , we predict the target variable with accuracy metric of AUC of 0.99.



This high overfit could be due to the sampling technique, however, given our recall is an important metric, we would be more concerned with predicting patients with strokes than the others which may justify our choice of sampling even under overfit. This overfit may predict the non-stroke people incorrectly which we may still afford given the ones with stroke are all correctly predicted.

Save the File

Finally, I saved the file and upload it in SSH. I used the library function PySPark2PMML.