Big Data Systems and Techniques

Classification in a big data environment

Dr. Dinos Arkoumanis - arkoumanis.dinos@gmail.com

Panagiotis Rallis
ID:P3351816

Task 1 - Get the data

Firstly we download the products sql:

```
cd /opt/
#Download file
wget --user dinospublic@yahoo.gr --password forsharingpurposes
https://bitbucket.org/dinosar/bigdatasystemscourse/downloads/products.s
ql.zip

#unzip
unzip products.sql.zip

#start prostgress
sudo -u postgres psql
```

Create database products_project using the products.sql

```
# create database and table temp_products from products.sql
CREATE USER prallis_ds WITH PASSWORD '13131966';
CREATE DATABASE products_project;
#connect to product_project database
\c products_project;
```

Create the schema of database and insert the data according to products.sql

```
\i /opt/products.sql
```

Give GRANT privileges to user prallis_ds

```
# Privileges
\pset tuples_only on
\o /tmp/grant-privs2
SELECT 'GRANT SELECT,INSERT,UPDATE,DELETE ON "'  || schemaname || '".
"' ||tablename ||'" TO prallis_ds;' FROM pg_tables WHERE tableowner =
CURRENT_USER and schemaname = 'public';
\o
\pset tuples_only off
```

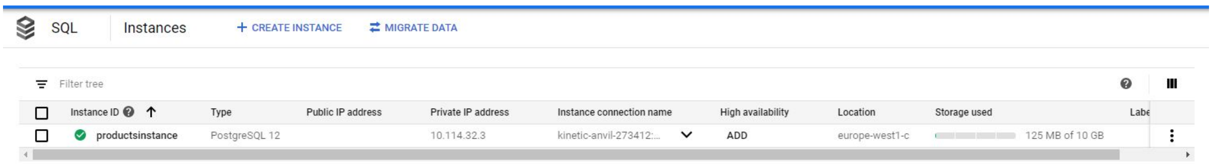
```
\i /tmp/grant-privs2
\q
```

```
products=# select count(*) from temp_products;
```

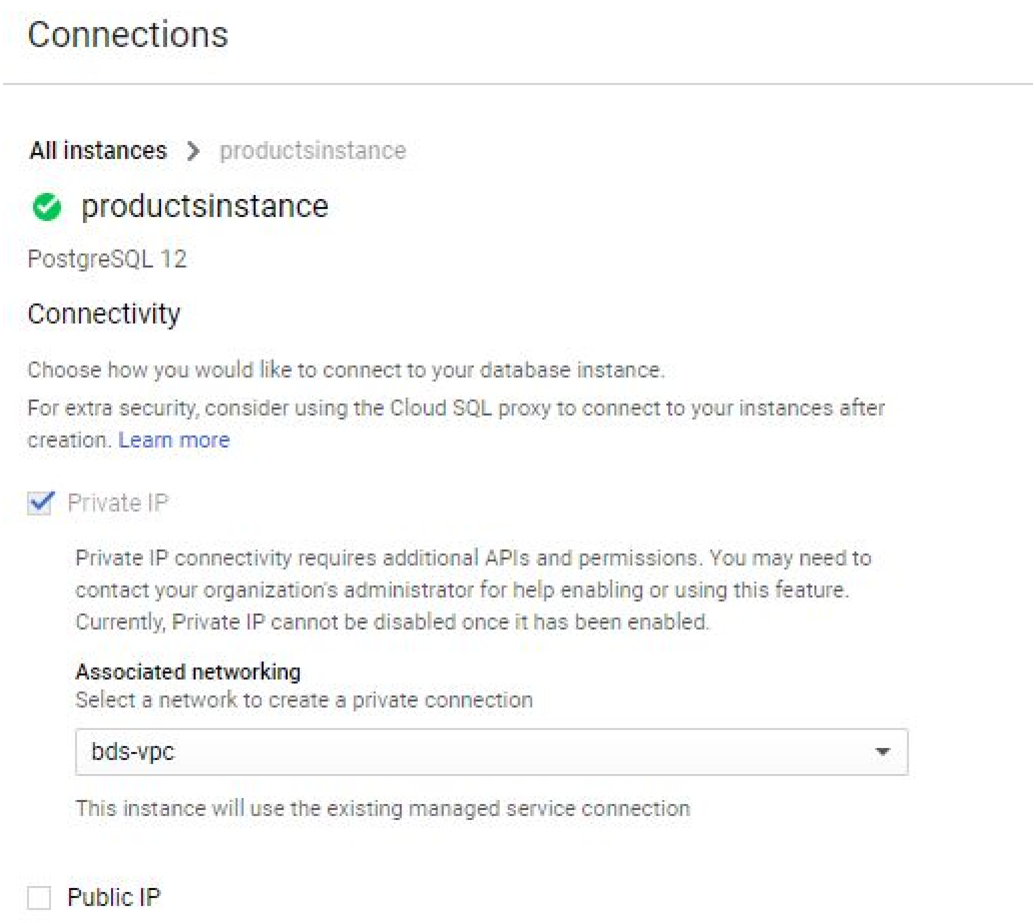
```
postgres=# \c products_project
You are now connected to database "products_project" as user "postgres".
products_project=# select count(*) from temp_products;
  count
---------
 2080734
(1 row)

products_project=#
```

Postgres Google Cloud Instance

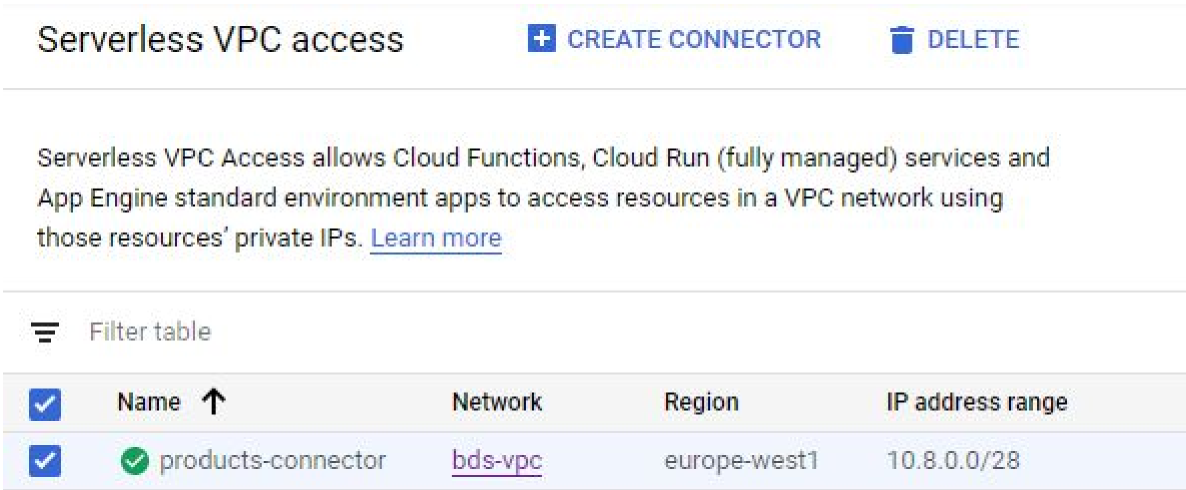
Firstly we create an SQL Google Instance of type Postgresql 12



with private connections to the bastion virtual private network



Secondly we create a serverless vpc access



I am not sure if the previous step is needed.

At the node s01

```
sudo apt-get update
sudo apt-get install postgresql-client
```

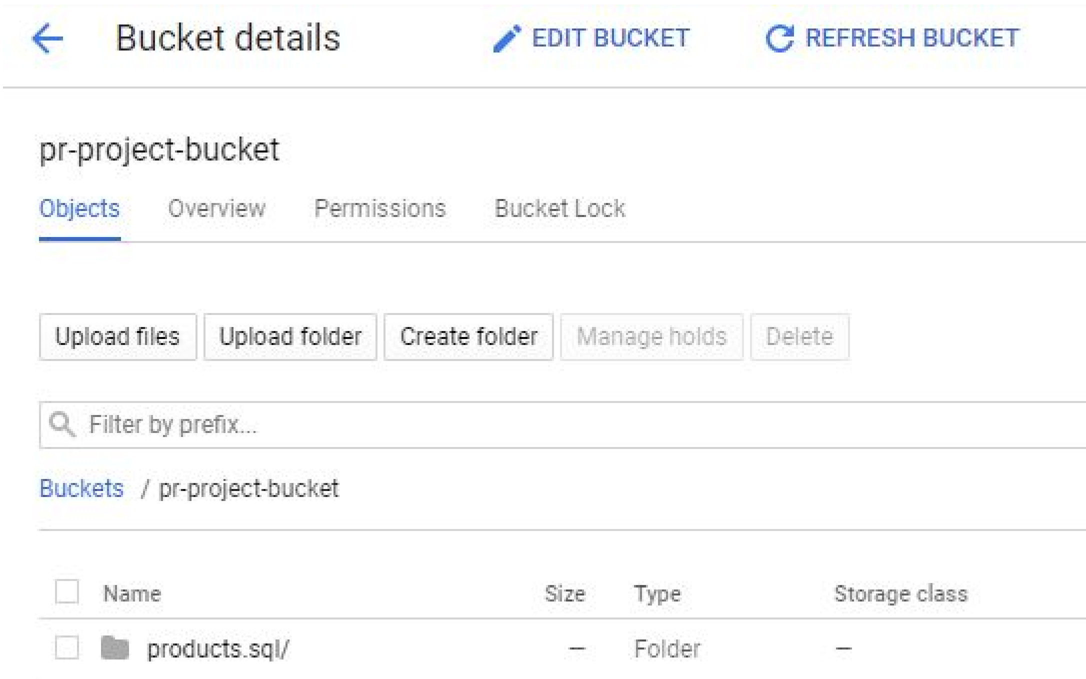
```
psql -h 10.114.32.3 -U postgres
```

```
10.114.32.3 is the private IP of Google Cloud SQL products instance
```

Therefore our connection from the s01 compute engine to the Cloud postgres is ready.

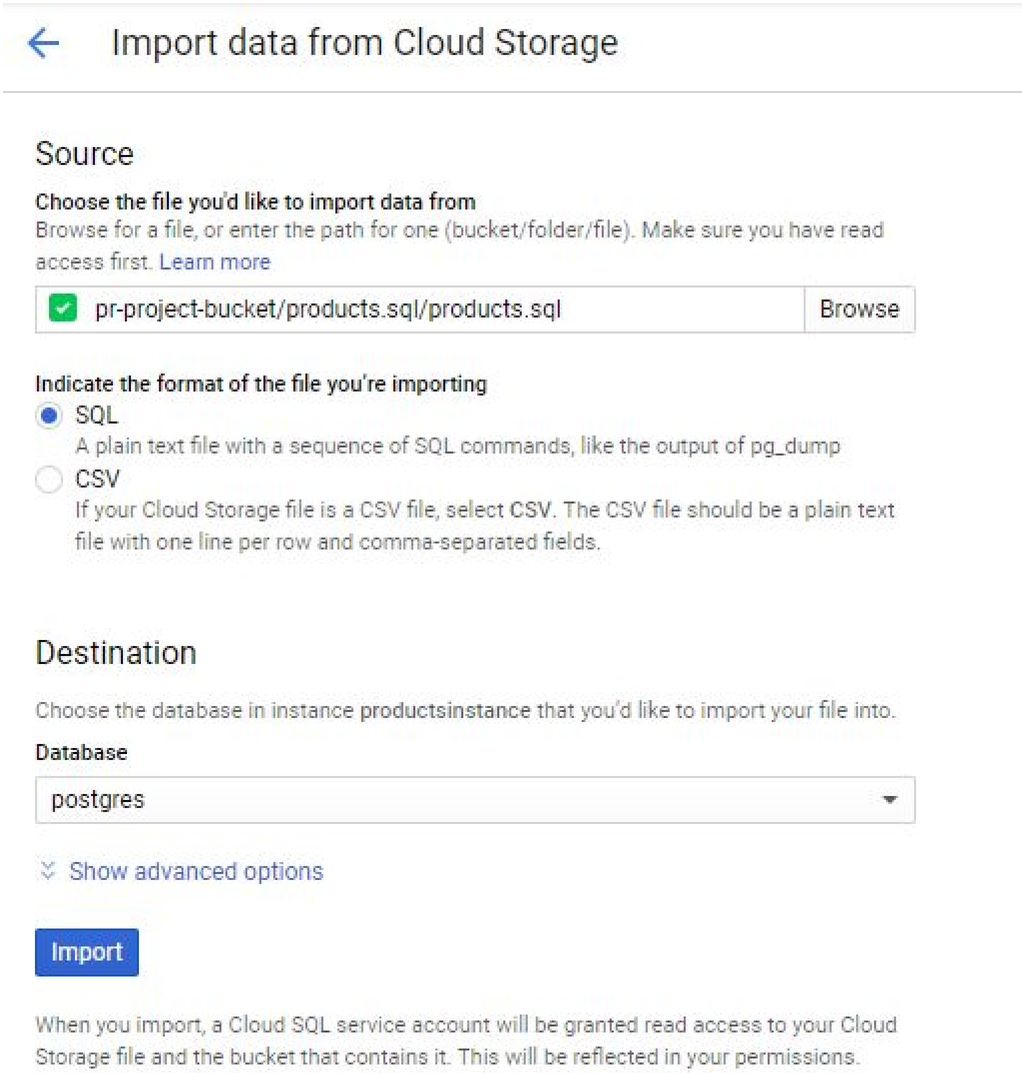
Now, we have to import the data to the cloud database.

We create a bucket



where we upload the sql file products.sql.

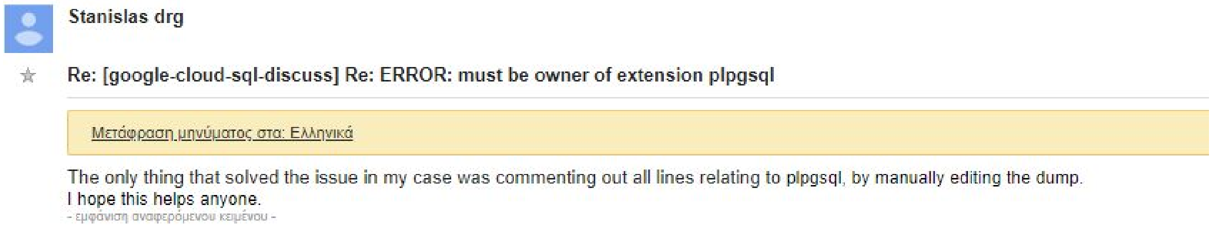
And finally, import data from the bucket to the postgres instance



However, I have a problem with the permissions during the import of products.sql to the Postgres Instance

Type	Status
Import	exit status 3 SET SET SET SET SET SET CREATE EXTENSION ERROR: must be owner of extension plpgsql

Looking for a solution, I found this


Stanislas drg

Re: [google-cloud-sql-discuss] Re: ERROR: must be owner of extension plpgsql

Μετάφραση μηνύματος στα: Ελληνικά

The only thing that solved the issue in my case was commenting out all lines relating to plpgsql, by manually editing the dump.
I hope this helps anyone.
- εμφάνιση αναφερόμενου κειμένου -

While I cannot open the products.sql file to comment the plpgsql lines, I will continue my implementation using the manual creation of the database.

Alternative we could create our database products using the file products.sql, via s01 node which is connected to the cloud postgres.

Task 2 - Create a parquet file

Locate the categories of shoes and a create a subset of data with shoes only.

```
products_project=#
select distinct category_id,category_code, category_name from
temp_products where category_code
like ('%shoe%') or category_code like '%sandal%' or category_code like
'%boots%' or category_code like '%flats%' or
category_code like '%sneak%' or category_code like '%clog%' or
category_code like '%slip%' or category_code like '
%heel%' or category_code like 'loafer' or category_code like '%footw%'
or name like '%footw%'  order by 3;
```

```
 category_id |    category_code     |   category_name
-------------+----------------------+--------------------
        1976 | shoes-athletic       | Athletic Shoes
        1561 | mens-shoes-athletic  | Athletic Shoes
        1069 | boots                | Boots
        1562 | mens-boots           | Boots
        1599 | boys-shoes           | Boys' Shoes
        1773 | evening-shoes        | Evening Shoes
        1070 | flats                | Flats
        1612 | girls-shoes          | Girls' Shoes
        1564 | mens-lace-up-shoes   | Lace-up Shoes
        1948 | mules-and-clogs      | Mules & Clogs
        1970 | sandals              | Sandals
        1563 | mens-sandals         | Sandals
        1386 | bridal-shoes         | Shoes
        1565 | mens-slip-ons-shoes  | Slip-ons & Loafers
        1931 | mens-slippers        | Slippers
        1412 | slippers             | Slippers
        1566 | mens-sneakers        | Sneakers
        2008 | womens-sneakers      | Sneakers
(18 rows)
```

Group by per category_code and count to find the most popular shoes categories.

```
select category_code,count(*) from temp_products group by category_code
order by count desc limit 20;
```

```
products_project=# select category_code,count(*) from temp_products group by category_code order by count desc limi
t 20;
     category_code      | count
------------------------+-------
 boots                  | 61412
 sandals                | 44405
 day-dresses            | 38916
 necklaces              | 38267
 earrings               | 36601
 longsleeve-tops        | 34074
 mens-tees-and-tshirts  | 33650
 watches                | 32439
 bracelets              | 32196
 mens-longsleeve-shirts | 31658
 flats                  | 30240
 pumps                  | 30083
 girls-shoes            | 21632
 sunglasses             | 21379
 mens-watches           | 20714
 cocktail-dresses       | 20582
 casual-pants           | 19382
 shoulder-bags          | 18894
 shortsleeve-tops       | 18667
 mens-sneakers          | 18560
(20 rows)
```

```
CREATE TABLE shoes as select * from temp_products where category_code
in ('boots','sandals','girls-shoes','mens-sneakers');
SELECT 146009
```

Table shoes

```
 product_id |         name          | upc_id |                          descr                          |
                 vendor_catalog_url                                 |                                                  buy_url
                                      | manufacturer_name | sale_price | retail_price | manufacturer_part_no | co
untry | vendor_id | category_name | category_code | category_id
------------+----------------------+--------+---------------------------------------------------------+------------
------------------------------------------------------------------+-----------------------------------------------
--------------------------------------+-------------------+------------+--------------+----------------------+---
------+-----------+---------------+---------------+-------------
     444554 | Mou 'Eskimo 50' boots |        | Black sheep skin and wool 'Eskimo 50' boots from Mou. | http://www.s
hopstyle.com/p/mou-eskimo-50-boots/458491790?pid=uid8576-26123524-6 | http://www.shopstyle.com/action/apiVisitRetai
ler?id=458491790&pid=uid8576-26123524-6 | Mou               |     462.72 |       462.72 |                      |
      |           | Boots         | boots         |        1069
(1 row)
```

Write a spark program that connects to postgresdb and reads the data in DataFrame

Connect to Jupyter Notebook

```
sh start-notebook.sh
./ngrok http 8880
```

spark program

```
from pyspark.sql import SparkSession
from pyspark.sql import Row

df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:postgresql://s01:5432/products_project") \
    .option("driver","org.postgresql.Driver") \
    .option("dbtable", "(select * from shoes ) as shoes") \
    .option("user", "prallis_ds") \
    .option("password", "13131966") \
    .load()

df.printSchema()
df.show(10)
```

```
root
 |-- product_id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- upc_id: string (nullable = true)
 |-- descr: string (nullable = true)
 |-- vendor_catalog_url: string (nullable = true)
 |-- buy_url: string (nullable = true)
 |-- manufacturer_name: string (nullable = true)
 |-- sale_price: decimal(38,18) (nullable = true)
 |-- retail_price: decimal(38,18) (nullable = true)
 |-- manufacturer_part_no: string (nullable = true)
 |-- country: string (nullable = true)
 |-- vendor_id: integer (nullable = true)
 |-- category_name: string (nullable = true)
 |-- category_code: string (nullable = true)
 |-- category_id: integer (nullable = true)
```

```
+----------+------------------+------+------------------+------------------+------------------+----------------+------
-------------+------------------+------------------+------+--------+------------+------------+----------+
|product_id|              name|upc_id|             descr|  vendor_catalog_url|           buy_url|manufacturer_name|
sale_price|      retail_price|manufacturer_part_no|country|vendor_id|category_name|category_code|category_id|
+----------+------------------+------+------------------+------------------+------------------+----------------+------
-------------+------------------+------------------+------+--------+------------+------------+----------+
|    444554|Mou 'Eskimo 50' b...|  null|Black sheep skin ...|http://www.shopst...|http://www.shopst...|           Mou|462.72
00000000000...|462.7200000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    449588|Dr. Martens '1460...|  null|Black leather '14...|http://www.shopst...|http://www.shopst...|     Dr. Martens|168.16
00000000000...|168.1600000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    441781|La Canadienne Wom...|  null|La Canadienne Wom...|http://www.shopst...|http://www.shopst...|   La Canadienne|249.95
00000000000...|249.9500000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442197| LE SILLA Tall boots|  null|Zip closure<br>Ro...|http://www.shopst...|http://www.shopst...|       Le Silla|1210.0
00000000000...|1210.000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442199|JIL SANDER Ankle ...|  null|Round toeline<br>...|http://www.shopst...|http://www.shopst...|     Jil Sander|995.00
00000000000...|995.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442254|   Jenni Kayne Boots|  null|Black Jenny Kayne...|http://www.shopst...|http://www.shopst...|    Jenni Kayne|115.00
00000000000...|115.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442255|    Jimmy Choo Boots|  null|Tan Jimmy Choo bo...|http://www.shopst...|http://www.shopst...|     Jimmy Choo|235.00
00000000000...|235.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442256|    Jimmy Choo Boots|  null|Black Jimmy Choo ...|http://www.shopst...|http://www.shopst...|     Jimmy Choo|260.00
00000000000...|260.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    442257|   Gucci Ankle Boots|  null|Black Gucci suede...|http://www.shopst...|http://www.shopst...|          Gucci|295.00
00000000000...|295.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
|    444723|VIC MATIE' Over t...|  null|Zip closure<br>Ro...|http://www.shopst...|http://www.shopst...|      Vic Mati�|965.00
00000000000...|965.0000000000000...|                null|   null|     null|       Boots|        boots|      1069|
+----------+------------------+------+------------------+------------------+------------------+----------------+------
-------------+------------------+------------------+------+--------+------------+------------+----------+
only showing top 10 rows
```

Write the DataFrame in HDFS in parquet format

Create folder products_project in Hadoop

```
$ hdfs dfs -ls /
```

```
$ hdfs dfs -mkdir /products_project
```

Create the parquet file from jupyter

```
df.write.parquet("hdfs:///products_project/shoes.parquet")
```

```
hdfs dfs -ls /products_project
```

```
root@s01:~# hdfs dfs -ls /products_project
Found 1 items
drwxr-xr-x   - root supergroup          0 2020-06-26 15:35 /products_project/shoes.parquet
root@s01:~#
```

Task 3 - ML

Logistic Regression

Load shoes parquet from Hadoop

```
shoes_parquet = spark.read.parquet("hdfs:///products_project/shoes.parquet")
```

Select descr and category code to process for the training.

```
data = shoes_parquet.select('descr','category_code')
data.show(5)
```

```
+--------------------+-------------+
|               descr|category_code|
+--------------------+-------------+
|Black sheep skin ...|        boots|
|Black leather '14...|        boots|
|La Canadienne Wom...|        boots|
|Zip closure<br>Ro...|        boots|
|Round toeline<br>...|        boots|
+--------------------+-------------+
only showing top 5 rows
```

Data Preprocess

Firstly we transform the category code from string to integer, in order to used as Label in training process.

At the description, we apply a tokenizer to separate each description to words.

The output of the tokenizer is feeded to tha hashing tf and the output of hashing tf to the IDF, creating the features of TF-IDF scores for each word.

Finally, we split our data to train and test with 70-30 separation percentage.

```
from pyspark.ml.classification import LogisticRegression, OneVsRest
```

```
# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.
labelIndexer = StringIndexer(inputCol="category_code", outputCol="label").fit(data)

tokenizer = Tokenizer(inputCol="descr", outputCol="words")
wordsData = tokenizer.transform(data)

# Automatically identify categorical features, and index them.
# hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="rawFeatures")
featurizedData = hashingTF.transform(wordsData)

idf = IDF(inputCol="rawFeatures", outputCol="features")

(train, test) = data.randomSplit([0.7, 0.3])
train.cache()
```

Our pipeline contains the above implementations and the One vs Rest Logistic Regression Classifier.

```
# instantiate the base classifier.
lr = LogisticRegression(maxIter=10, tol=1E-6, fitIntercept=True)
```

```
# instantiate the One Vs Rest Classifier.
ovr = OneVsRest(classifier=lr)
```

```
pipeline = Pipeline(stages=[labelIndexer, tokenizer, hashingTF,idf , ovr])
```

We use the Cross Validation Verbose class of Course 7 Machine Learning Pipelines with the created pipeline.

```
paramGrid = ParamGridBuilder() \
    .addGrid(hashingTF.numFeatures, [10, 1000, 10000]) \
    .build()
crossval = CrossValidatorVerbose(estimator=pipeline,
                     estimatorParamMaps=paramGrid,
                     evaluator=MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accu
                     numFolds=5)
```

We train our model using the train data set.

```
# train the multiclass model.
ovrModel = crossval.fit(train)
```

```
Comparing models on fold 1
params: {'numFeatures': 10}     accuracy: 0.463998      avg: 0.463998
params: {'numFeatures': 1000}   accuracy: 0.897535      avg: 0.897535
params: {'numFeatures': 10000}  accuracy: 0.946156      avg: 0.946156
Comparing models on fold 2
params: {'numFeatures': 10}     accuracy: 0.457295      avg: 0.460646
params: {'numFeatures': 1000}   accuracy: 0.894005      avg: 0.895770
params: {'numFeatures': 10000}  accuracy: 0.945710      avg: 0.945933
Comparing models on fold 3
params: {'numFeatures': 10}     accuracy: 0.462781      avg: 0.461358
params: {'numFeatures': 1000}   accuracy: 0.896687      avg: 0.896076
params: {'numFeatures': 10000}  accuracy: 0.948515      avg: 0.946793
Comparing models on fold 4
params: {'numFeatures': 10}     accuracy: 0.463759      avg: 0.461958
params: {'numFeatures': 1000}   accuracy: 0.898209      avg: 0.896609
params: {'numFeatures': 10000}  accuracy: 0.947994      avg: 0.947094
Comparing models on fold 5
params: {'numFeatures': 10}     accuracy: 0.461410      avg: 0.461849
params: {'numFeatures': 1000}   accuracy: 0.896718      avg: 0.896631
params: {'numFeatures': 10000}  accuracy: 0.945783      avg: 0.946831
Best model:
params: {'numFeatures': 10000}  accuracy: 0.946831
```

As we can observer the Logistic Regression Classifier achieves a high accuracy in training data.

Also the accuracy of the model in test data is around 94%.

```
# score the model on test data.
predictions = ovrModel.transform(test)

# obtain evaluator.
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")

# compute the classification error on test data.
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g" % (1.0 - accuracy))
```

```
Test Error = 0.0511128
```

Save the best model of cross validation in hadoop

```
lr_model = ovrModel.bestModel
```

```
outPath = "hdfs:///products_project/model"
```

```
#Save model
lr_model.save(outPath)
```

```
root@s01:~# hdfs dfs -ls /products_project/model
Found 2 items
drwxr-xr-x   - root supergroup          0 2020-07-02 18:08 /products_project/model/metadata
drwxr-xr-x   - root supergroup          0 2020-07-02 18:08 /products_project/model/stages
```

Random Forest Classifier

The same approach for the Random Forest Classifier with one extra parameter on grid search about the number of trees.

```
# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.
labelIndexer = StringIndexer(inputCol="category_code", outputCol="indexedLabel").fit(data)

tokenizer = Tokenizer(inputCol="descr", outputCol="words")
# wordsData = tokenizer.transform(data)

# Automatically identify categorical features, and index them.
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="rawFeatures")
# featurizedData = hashingTF.transform(wordsData)

idf = IDF(inputCol="rawFeatures", outputCol="features")

labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol='features', numTrees=10)

pipeline = Pipeline(stages=[labelIndexer, tokenizer, hashingTF, idf,  rf, labelConverter])

(train, test) = data.randomSplit([0.7, 0.3])
train.cache()

paramGrid = ParamGridBuilder() \
    .addGrid(hashingTF.numFeatures, [10, 1000, 10000]) \
    .addGrid(rf.numTrees, [10, 20]) \
    .build()

crossval = CrossValidatorVerbose(estimator=pipeline,
                      estimatorParamMaps=paramGrid,
                      evaluator=MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricNam
                      numFolds=5)

rf_model = crossval.fit(train)
```

```
Comparing models on fold 1
params: {'numTrees': 10, 'numFeatures': 10}     accuracy: 0.476558      avg: 0.476558
params: {'numTrees': 20, 'numFeatures': 10}     accuracy: 0.477826      avg: 0.477826
params: {'numTrees': 10, 'numFeatures': 1000}   accuracy: 0.579743      avg: 0.579743
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.618822      avg: 0.618822
params: {'numTrees': 10, 'numFeatures': 10000}  accuracy: 0.476509      avg: 0.476509
params: {'numTrees': 20, 'numFeatures': 10000}  accuracy: 0.490218      avg: 0.490218
Comparing models on fold 2
params: {'numTrees': 10, 'numFeatures': 10}     accuracy: 0.469504      avg: 0.473031
params: {'numTrees': 20, 'numFeatures': 10}     accuracy: 0.470187      avg: 0.474006
params: {'numTrees': 10, 'numFeatures': 1000}   accuracy: 0.583979      avg: 0.581861
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.628931      avg: 0.623877
params: {'numTrees': 10, 'numFeatures': 10000}  accuracy: 0.476720      avg: 0.476614
params: {'numTrees': 20, 'numFeatures': 10000}  accuracy: 0.481351      avg: 0.485785
Comparing models on fold 3
params: {'numTrees': 10, 'numFeatures': 10}     accuracy: 0.465119      avg: 0.470393
params: {'numTrees': 20, 'numFeatures': 10}     accuracy: 0.472696      avg: 0.473570
params: {'numTrees': 10, 'numFeatures': 1000}   accuracy: 0.599316      avg: 0.587679
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.625813      avg: 0.624522
params: {'numTrees': 10, 'numFeatures': 10000}  accuracy: 0.481203      avg: 0.478144
params: {'numTrees': 20, 'numFeatures': 10000}  accuracy: 0.458372      avg: 0.476647
Comparing models on fold 4
params: {'numTrees': 10, 'numFeatures': 10}     accuracy: 0.471632      avg: 0.470703
params: {'numTrees': 20, 'numFeatures': 10}     accuracy: 0.470696      avg: 0.472851
params: {'numTrees': 10, 'numFeatures': 1000}   accuracy: 0.581653      avg: 0.586173
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.583231      avg: 0.614199
params: {'numTrees': 10, 'numFeatures': 10000}  accuracy: 0.465076      avg: 0.474877
params: {'numTrees': 20, 'numFeatures': 10000}  accuracy: 0.472864      avg: 0.475701
Comparing models on fold 5
params: {'numTrees': 10, 'numFeatures': 10}     accuracy: 0.467664      avg: 0.470095
params: {'numTrees': 20, 'numFeatures': 10}     accuracy: 0.470606      avg: 0.472402
params: {'numTrees': 10, 'numFeatures': 1000}   accuracy: 0.596764      avg: 0.588291
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.630596      avg: 0.617478
params: {'numTrees': 10, 'numFeatures': 10000}  accuracy: 0.469919      avg: 0.473885
params: {'numTrees': 20, 'numFeatures': 10000}  accuracy: 0.464133      avg: 0.473388
Best model:
params: {'numTrees': 20, 'numFeatures': 1000}   accuracy: 0.617478
```

```
prediction = rf_model.transform(test)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
```

```
accuracy = evaluator.evaluate(prediction)
print("Test Error = %g " % (1.0 - accuracy))
```

```
Test Error = 0.414736
```

It is clear that the Random Forest Classifier was not able to achieve a decent accuracy as the Logistic Regression. Therefore we will handle the prediction of tweets with the logistic model which was saved on hadoop.

Task 4 - Kafka

The below twitter.py program connects to the Twitter API with my tokens and filtering the tweets with the words shoes offers, boots offers, sneakers offers, girls shoes offers, the hashtag #shoesoffers and sends the produced tweets to the offer topic.

```
from __future__ import print_function

import json

from kafka import KafkaProducer, KafkaClient

import tweepy

access_token = "1276871198543093763-gBx3jtk56OWNFk9cjTbKnKmcUBkDgj"

access_token_secret =  "7hckEvlAdZTeAEDkkn9guQJ4DvFdjOLXy50XHbrq2HJSL"

consumer_key =  "NRqLKTJB9LFcO1ztoKOafwRMk"

consumer_secret =  "6H3U3y7gG1yropk4sHej7g0Y9j21BKoAaiPy2XLHKYkLX1Qj05"

# Words to track

WORDS = ['shoes offers', 'boots offers','sneakers offers','girls shoes 
offers' '#shoesoffers']

class StreamListener(tweepy.StreamListener):

    # This is a class provided by tweepy to access the Twitter 
Streaming API.

    def on_connect(self):

        # Called initially to connect to the Streaming API

        print("You are now connected to the streaming API.")

    def on_error(self, status_code):

        # On error - if an error occurs, display the error / status 
code

        print("Error received in kafka producer " + repr(status_code))

        return True # Don't kill the stream

    def on_data(self, data):
```

```
        """ This method is called whenever new data arrives from live
stream.

        We asynchronously push this data to kafka queue"""

        try:

            producer.send('offers', data.encode('utf-8'))

        except Exception as e:

            print(e)

            return False

        return True # Don't kill the stream


    def on_timeout(self):

        return True # Don't kill the stream
```

In order to watch the collected tweets, we need to create the ZooKeeper synchronization service and the kafka consumer with the topic offers.

```
./start.sh
```

check if zookeeper and kafka server started

open new MASTER s01 a

```
cd /opt/kafka_2.11-0.10.1.0
```

create zookeeper and topic offers

```
bin/kafka-topics.sh --create --zookeeper localhost:2181
--replication-factor 1 --partitions 1 --topic offers
```

open new MASTER s01 b

```
cd /opt/kafka_2.11-0.10.1.0
```

run consumer

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic offers
```

```
root@s01:/opt/kafka_2.11-0.10.1.0# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic offers
Using the ConsoleConsumer with old consumer is deprecated and will be removed in a future major release. Consider u
sing the new consumer by passing [bootstrap-server] instead of [zookeeper].
```

open new MASTER s01 c

```
python twitter.py
```

```
root@s01:~# python twitter.py
Tracking: ['shoes offers', 'boots offers', 'sneakers offers', 'girls shoes offers', '#shoesoffers']
You are now connected to the streaming API.
```

the streaming tweets are printed at the s01 b consumer.


```
{"created_at":"Sun Jun 28 13:51:17 +0000
2020","id":1277238139778748416,"id_str":"1277238139778748416","text":"shoes offers shopping offers
shoes #shoesoffers  Dinos vale ena 10 na pame gia kana mpanio","source":"\u003ca
href=\"https:\/\/mobile.t
witter.com\" rel=\"nofollow\"\u003eTwitter Web
App\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,
"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"i
d":1276871198543093763,"id_str":"1276871198543093763","name":"Panagiotis
Rallis","screen_name":"PanagiotisRall3","location":null,"url":null,"description":null,"translator_ty
pe":"none","protected":false,"verified":false,"followers_count":0,"friend
s_count":0,"listed_count":0,"favourites_count":0,"statuses_count":2,"created_at":"Sat Jun 27
13:33:22 +0000
2020","utc_offset":null,"time_zone":null,"geo_enabled":false,"lang":null,"contributors_enabled":fals
e,"is_translator":false,"profile_backg
round_color":"F5F8FA","profile_background_image_url":"","profile_background_image_url_https":"","pro
file_background_tile":false,"profile_link_color":"1DA1F2","profile_sidebar_border_color":"C0DEED","p
rofile_sidebar_fill_color":"DDEEF6","profile_t
ext_color":"333333","profile_use_background_image":true,"profile_image_url":"http:\/\/abs.twimg.com\
/sticky\/default_profile_images\/default_profile_normal.png","profile_image_url_https":"https:\/\/ab
s.twimg.com\/sticky\/default_profile_images\/d
efault_profile_normal.png","default_profile":true,"default_profile_image":false,"following":null,"fo
llow_request_sent":null,"notifications":null},"geo":null,"coordinates":null,"place":null,"contributo
rs":null,"is_quote_status":false,"quote_count"
:0,"reply_count":0,"retweet_count":0,"favorite_count":0,"entities":{"hashtags":[{"text":"shoesoffers
","indices":[35,47]}],"urls":[],"user_mentions":[],"symbols":[]},"favorited":false,"retweeted":false
,"filter_level":"low","lang":"en","timestamp_m
s":"1593352277117"}
```

Task 5 - Spark streaming

While the consumer and the twitter streaming API are up and running, from the jupyter notebook we run the below spark script which

- **consumes from Kafka topic offers with Spark Streaming**
- **process the tweets to keep only the text json object**
- **save the streamed clean tweets to hadoop**

```
from pyspark.streaming.kafka import KafkaUtils, TopicAndPartition
from pyspark.streaming import StreamingContext
from __future__ import print_function

import sys
import json
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils


ssc = StreamingContext(sc, 1)
topicPartion = TopicAndPartition('offers',0)
topic = 'offers'
fromOffset = {topicPartion: 0}

twitterKafkaStream = KafkaUtils.createDirectStream(ssc, [topic],{"bootstrap.servers": 'localhost:9092'}, fromOffsets=fromOffset)

tweets = twitterKafkaStream. \
        map(lambda (key, value): json.loads(value)). \
        map(lambda json_object: (json_object["text"]))

tweets.saveAsTextFiles('/tweets/')
tweets.pprint(10)


ssc.start()
ssc.awaitTermination()
```

```
-------------------------------------------
Time: 2020-07-04 12:32:36
-------------------------------------------
here you can find sneakers offers #shoesoffers
the best shoes offers for ladies #shoesoffers
boots offers in summer why not

-------------------------------------------
Time: 2020-07-04 12:32:37
-------------------------------------------

-------------------------------------------
Time: 2020-07-04 12:32:38
-------------------------------------------

-------------------------------------------
Time: 2020-07-04 12:32:39
-------------------------------------------
```

```
root@s01:~# hdfs dfs -ls /tweets/
Found 106 items
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865956000
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865957000
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865958000
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865959000
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865960000
drwxr-xr-x   - root supergroup          0 2020-07-04 12:32 /tweets/-1593865961000
```

Load the new tweets with spark from the saved file on hadoop

```
: new_tweets = spark.read.csv("hdfs:///tweets/-1593865956000/part-00000")
```

```
: new_tweets.show(10)
```

```
+--------------------+
|                 _c0|
+--------------------+
|here you can find...|
|the best shoes of...|
|boots offers in s...|
+--------------------+
```

Rename the column name to descr

```
oldColumns = new_tweets.schema.names
newColumns = ["descr"]

df = reduce(lambda new_tweets, idx: new_tweets.withColumnRenamed(oldColumns[idx], newColumns[idx]), xrange(len(oldColumns)), new_
df.printSchema()
df.show()
```

```
root
 |-- descr: string (nullable = true)

+--------------------+
|               descr|
+--------------------+
|here you can find...|
|the best shoes of...|
|boots offers in s...|
+--------------------+
```

Load the saved logistic pipeline model from hadoop

```
from pyspark.ml import PipelineModel
outPath = "hdfs:///products_project/model"
load_lr_model = PipelineModel.load(outPath)
```

Predict the category of each tweet using the loaded pipeline

```
predictions = load_lr_model.transform(df)
```

```
predictions.show(3)
```

```
+--------------------+--------------------+--------------------+--------------------+----------+
|               descr|               words|         rawFeatures|            features|prediction|
+--------------------+--------------------+--------------------+--------------------+----------+
|here you can find...|[here, you, can, ...|(10000,[1135,1425...|(10000,[1135,1425...|       3.0|
|the best shoes of...|[the, best, shoes...|(10000,[219,763,1...|(10000,[219,763,1...|       1.0|
|boots offers in s...|[boots, offers, i...|(10000,[1445,3965...|(10000,[1445,3965...|       0.0|
+--------------------+--------------------+--------------------+--------------------+----------+
```

```
+-------------+-----+
|category_code|label|
+-------------+-----+
|      sandals|  1.0|
+-------------+-----+
+-------------+-----+
|category_code|label|
+-------------+-----+
|        boots|  0.0|
+-------------+-----+
```

```
+-------------+-----+
|category_code|label|
+-------------+-----+
|mens-sneakers|  3.0|
+-------------+-----+
```

Write the tweets and their evaluation in a parquet file and in a text file

```
predictions.write.parquet("hdfs:///tweets/predictions/predictions.parquet")
```

```
import pyspark.sql.functions as F
def myConcat(*cols):
    concat_columns = []
    for c in cols[:-1]:
        concat_columns.append(F.coalesce(c, F.lit("*")))
        concat_columns.append(F.lit(" "))
    concat_columns.append(F.coalesce(cols[-1], F.lit("*")))
    return F.concat(*concat_columns)

df_text = predictions.withColumn("combined", myConcat(*df.columns)).select("combined")

df_text.show()

df_text.coalesce(1).write.format("text").option("header", "false").mode("append").save("hdfs:///tweets/predictions/predictions.t
```

```
+--------------------+
|            combined|
+--------------------+
|here you can find...|
|the best shoes of...|
|boots offers in s...|
+--------------------+
```

```
root@s01:~# hdfs dfs -mkdir /tweets/predictions
root@s01:~# hdfs dfs -ls /tweets/predictions
Found 2 items
drwxr-xr-x   - root supergroup          0 2020-07-04 13:15 /tweets/predictions/predictions.parquet
drwxr-xr-x   - root supergroup          0 2020-07-04 13:26 /tweets/predictions/predictions.txt
```