Amsterdam University
of Applied Sciences

SPARK-MLLIB VS. TENSORFLOW

Research Report

Data Engineer & Data Scientist

2019 – 2020

SPARK-MLLIB VS. TENSORFLOW

Research Report

AUTHOR
Yoshio Schermer (5007060587)

COURSE
Data Engineer & Data Scientist

DATE
6/7/2020

TYPE PROJECT
Research report

VERSION
1.0

Summary

In this report, a logistic regression model (built with Spark-MLlib) is compared to a neural network (built with TensorFlow). The comparison is done by classifying the sentiment of hotel reviews and obtaining the benchmarks from this.

The benchmark shows that the logistic regression model performs better than the neural network for a small subset of all the hotel reviews.

Also interesting is that the logistic regression model took longer to train than the neural network. Expected was that the logistic regression model would be trained faster considering that it's built on top of Spark. But this does not necessarily hold true.

Future research can look at how the models compare to one another on the same metrics for different sizes of the dataset.

Table of Contents

1. Introduction

This research report contains a comparison between two popular ML libraries, namely Spark-MLlib and TensorFlow. Spark-MLlib is an addition to Spark, which is an open source general-purpose distributed data processing framework (*MLlib | Apache Spark*, 2020). TensorFlow, on the other hand, is an open source ML platform for designing complex neural networks (*TensorFlow*, 2020).

Hypothesis
Knowing this, it is interesting to find out how Spark-MLlib matches up against TensorFlow. Therefore, I formulated the following hypothesis:

> *Neural networks (built with TensorFlow) perform better than logistic regression models (built with Spark-MLlib) in terms of basic metrics (accuracy, time to train, precision, recall, f1-score).*

Research question
The corresponding main and -sub research questions are as follows:

1. Does logistic regression in Spark MLlib perform better than a NN in TensorFlow for sentiment analysis in terms of basic metrics (accuracy, time to train, precision, recall, f1-score)?
    a. What is Spark-MLlib?
        i. What is Spark?
        ii. What is logistic regression?
    b. What is TensorFlow?
        i. What is a neural network?
    c. What are the advantages and disadvantages of each framework?
    d. How do the two frameworks benchmark against one another in terms of basic metrics (accuracy, time to train, precision, recall, and f1-score)?

Case Study
To answer the hypothesis, I'll be conducting a case study: sentiment analysis on hotel reviews. In this case study, we'll be using logistic regression from Spark-MLlib and a neural network designed in TensorFlow to perform the same ML task, namely sentiment analysis.

Data Sources
The dataset we'll be using can be found at https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe.

Reading Guide
The report follows the outline: background, case study, results, and conclusions.
In the first chapter, we'll give answers to the underlying sub research questions: a, b, and c (see above).
In the subsequent chapter, we'll do our case study, describing the architecture of the application and an in-depth analysis of the models (logistic regression, and neural networks in this case).
After this, in the chapter on results, we'll take a look at the benchmarks of the models from the case study.
And finally, we'll conclude with an answer to the research question and provide incentive for further research.

2. Background

In this chapter, we'll answer the underlying sub research questions of our main research question, namely:

- What is Spark-MLlib?
- What is TensorFlow?
- What are the advantages / disadvantages of each framework?

2.1 What is Spark-MLlib?

Before we dive into what Spark-MLlib is, we must look at what Spark in general is. After this, we'll look at MLlib and, in particular, logistic regression.

2.1.1 What is Spark and Spark-MLlib?

As mentioned in the introduction Spark is a general-purpose distributed data processing framework. This means that Spark is a framework that can use a cluster of computers to perform computations. This is particularly useful for large datasets, that would otherwise take forever to process on a single computer. Obviously, Big Data is a field where such datasets are. Unsurprisingly, where Big Data is, there is Machine Learning too. Hence, Spark is used heavily in this field. This is also the reason why Spark has an additional library on top of its core engine, MLlib (*Spark 101: What Is It, What It Does, and Why It Matters* | *MapR*, 2013). As can been seen in Figure 1, Spark has several libraries on top of its engine, including MLlib.

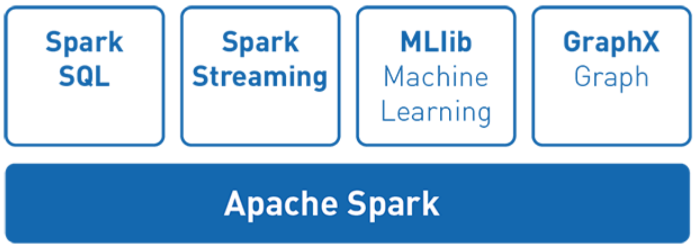


Figure 1. Spark and its libraries (2020). Mapr.Com. https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters/assets/image7.png.

2.1.2 What is logistic regression?

We all know that linear regression tries to find a linear model (a linear line) that best fits the given data. Logistic regression, on the other hand, tries to find a sigmoid function that best fits the data points with its probability and dependent variable. An example of a logistic regression curve can be seen in Figure 2. Here we see that the function outputs a higher probability given more hours of studying.

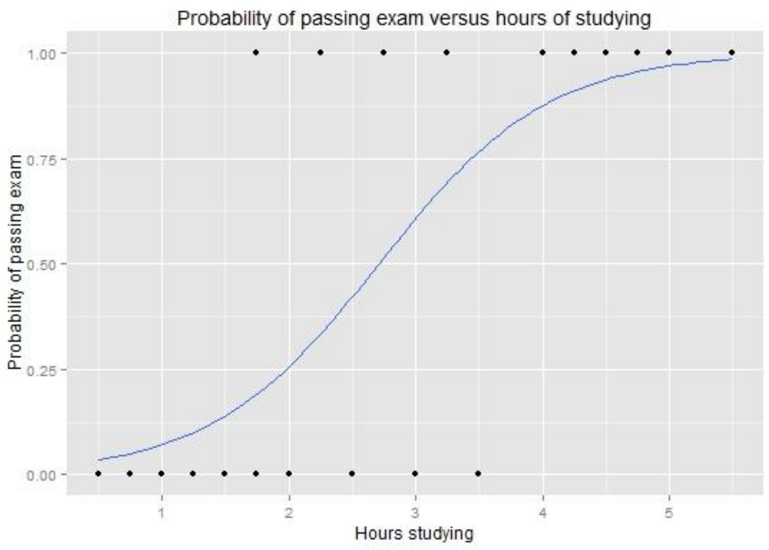


Figure 2. Graph of a logistic regression curve showing probability of passing an exam versus hours studying (G, 2015).

2.1.3 How does logistic regression differ in Spark-MLlib from other ML libraries?

Logistic regression can be found in many other popular ML libraries, such as in SciKit-Learn (*Sklearn.Linear_model.LogisticRegression — Scikit-Learn 0.23.1 Documentation*, 2014). Spark-MLlib, however, has an edge over other ML libraries when larger datasets have to be used.

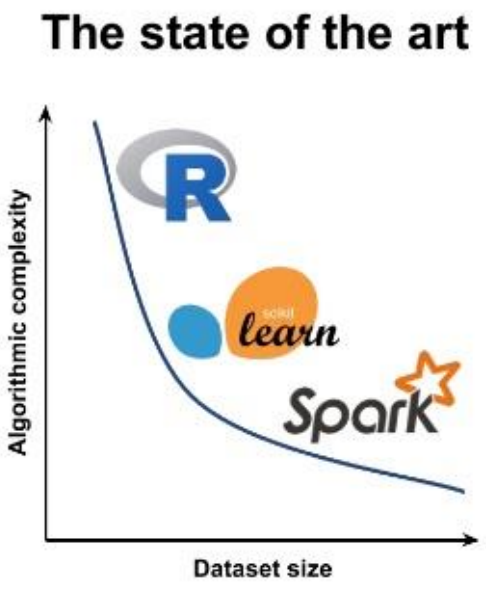


Figure 3. Comparing ML platforms with regards to complexity and dataset size. Spark becomes of greater use when datasets get larger (Villu Ruusmann, 2017).

2.2 What is TensorFlow?

In comparison to Spark or Spark-MLlib for that matter, TensorFlow is no data processing framework, nor is it built on top of one . TensorFlow is a standalone end-to-end open source machine learning platform developed by Google, meaning that you could use this platform / framework for building a ML model that has no intermediate steps such as in traditional ML (*TensorFlow*, 2020; *What Is End to End Learning in Machine Learning? - Quora*, 2019).
The framework is popular for designing complex neural networks as ML models (*TensorFlow*, 2020).
So, what are neural networks then?

2.3 What is a neural network?

A neural network is a ML model inspired by the human brain. Similar to the human brain, a neural network consists of neurons that are interconnected. In Figure 4, we see a general representation of what a neural network (not in the human brain) looks like. First, there is the input layer that takes in the features that are relevant for the machine to give the desired output. Between these layers are the hidden layers. As you can see from the same figure, the layers are entirely interconnected. This allows the model to find relationships between all of the given features as it passes forward (3Blue1Brown, 2020).

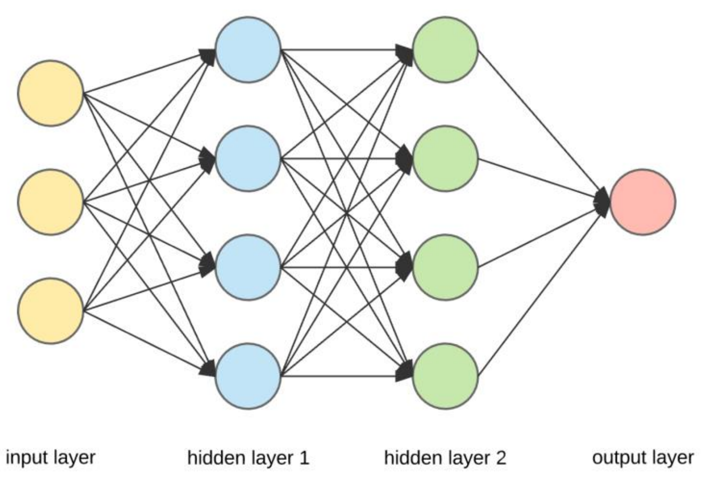


Figure 4. A general representation of a neural network (TowardsDataScience, 2020).

By also providing a loss function at the output layer, we're able to adjust the strength of the connections by using backpropagation. The eventual strengths trained determines what input results in what output (3Blue1Brown, 2020).

2.4 What are the advantages / disadvantages of each framework?

So, knowing these concepts. Why would we use one framework over the other? What are the pros and cons of each framework? Let's start with Spark-MLlib.

Spark-MLlib

Advantages of Spark-MLlib

- Built on top of Spark, which means that it can process large datasets efficiently (hence, faster).
- Easy to pick up if familiar with Scikit-Learn.
- Smaller community compared to TensorFlow.
- Open source.

- Free.
- Support for multiple programming languages: Scala, Java, Python, and R.

Disadvantages of Spark-MLlib

- Spark-MLlib has limited support for neural networks in comparison to TensorFlow (no deep learning).
- Confusing to work with RDDs, Spark DataFrames, Pandas DataFrames, Spark DataSets.
- Spark-MLlib is not the primary purpose of Spark.

How about TensorFlow?

TensorFlow

Advantages of TensorFlow

- Made for building complex neural networks (deep learning, which is the state-of-the-art in ML).
- Large community.
- TensorFlow has a clear, single primary purpose in comparison to Spark and its ML library.
- Open source.
- Free.

Disadvantages of TensorFlow

- Steep learning curve.

3. Case Study: Sentiment Analysis on Hotel Reviews

In this case study, we'll built an interactive dashboard showing useful descriptive analytics for hotel owners, so they can see their own ratings / reviews, but also of their competitors (see Figure x, x).

For analyzing the sentiment in the reviews, we'll use both logistic regression (from Spark-MLlib) and a neural network (built in TensorFlow). In the next chapter, we'll benchmark the results of both of these classifiers.

We'll first look at the general architecture of the application and follow up by going into depth with both models.

Data Source
The dataset we'll be using consists of 515k hotel reviews and can be found at https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe.

3.1 Architecture

For the general architecture see Figure 5. First, we bulk insert the data into MongoDB using Spark. Then, we clean the file using Spark DataFrames and insert this into a new MongoDB collection: clean. After this, we use this cleaned collection, again with Spark for model building for Spark-MLlib as well as for TensorFlow.

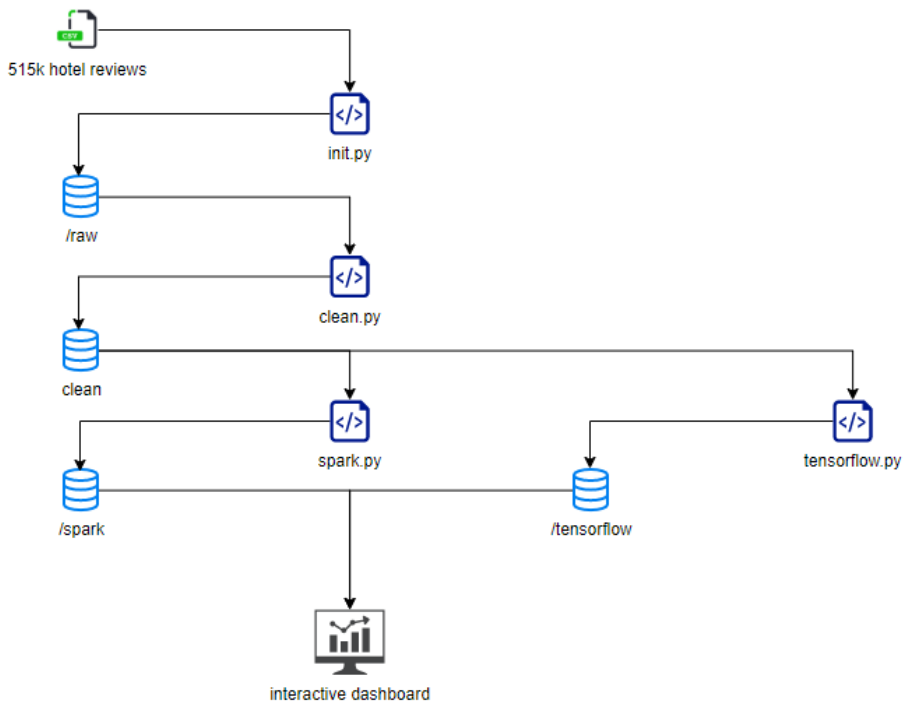
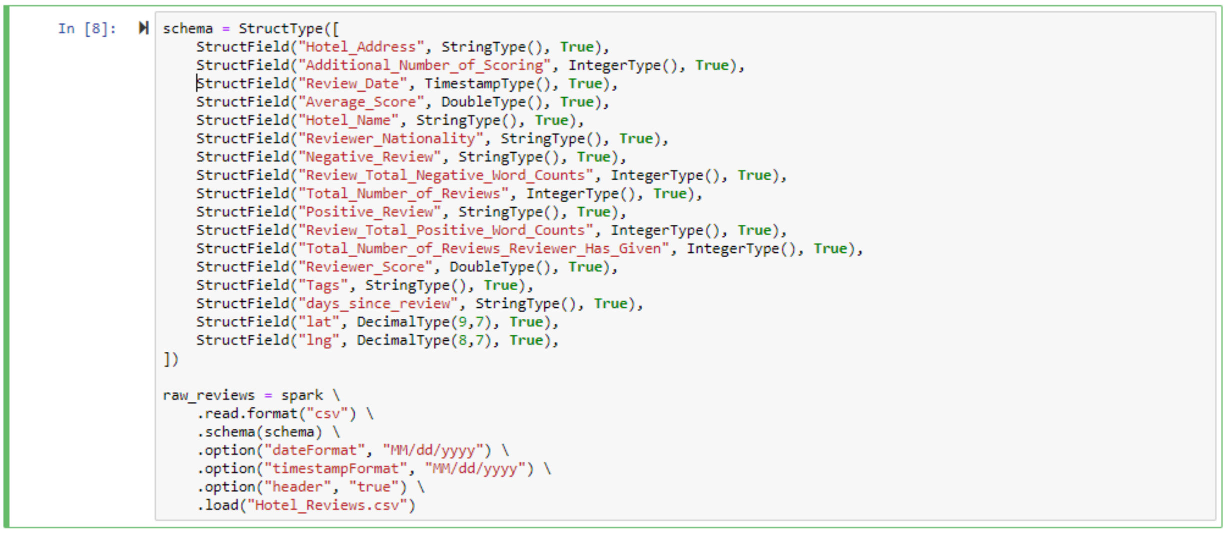


Figure 5. General architecture of application.

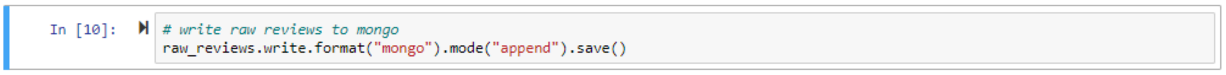
3.1.1 Data discovery

As mentioned just now, we read in the 515k hotel reviews from the previously mentioned CSV. Then, we bulk insert this into a MongoDB collection using Spark.



```
In [8]:  schema = StructType([
             StructField("Hotel_Address", StringType(), True),
             StructField("Additional_Number_of_Scoring", IntegerType(), True),
             StructField("Review_Date", TimestampType(), True),
             StructField("Average_Score", DoubleType(), True),
             StructField("Hotel_Name", StringType(), True),
             StructField("Reviewer_Nationality", StringType(), True),
             StructField("Negative_Review", StringType(), True),
             StructField("Review_Total_Negative_Word_Counts", IntegerType(), True),
             StructField("Total_Number_of_Reviews", IntegerType(), True),
             StructField("Positive_Review", StringType(), True),
             StructField("Review_Total_Positive_Word_Counts", IntegerType(), True),
             StructField("Total_Number_of_Reviews_Reviewer_Has_Given", IntegerType(), True),
             StructField("Reviewer_Score", DoubleType(), True),
             StructField("Tags", StringType(), True),
             StructField("days_since_review", StringType(), True),
             StructField("lat", DecimalType(9,7), True),
             StructField("lng", DecimalType(8,7), True),
         ])

         raw_reviews = spark \
             .read.format("csv") \
             .schema(schema) \
             .option("dateFormat", "MM/dd/yyyy") \
             .option("timestampFormat", "MM/dd/yyyy") \
             .option("header", "true") \
             .load("Hotel_Reviews.csv")
```

Figure 6. Reading the CSV file, and imposing a schema on it in Spark.

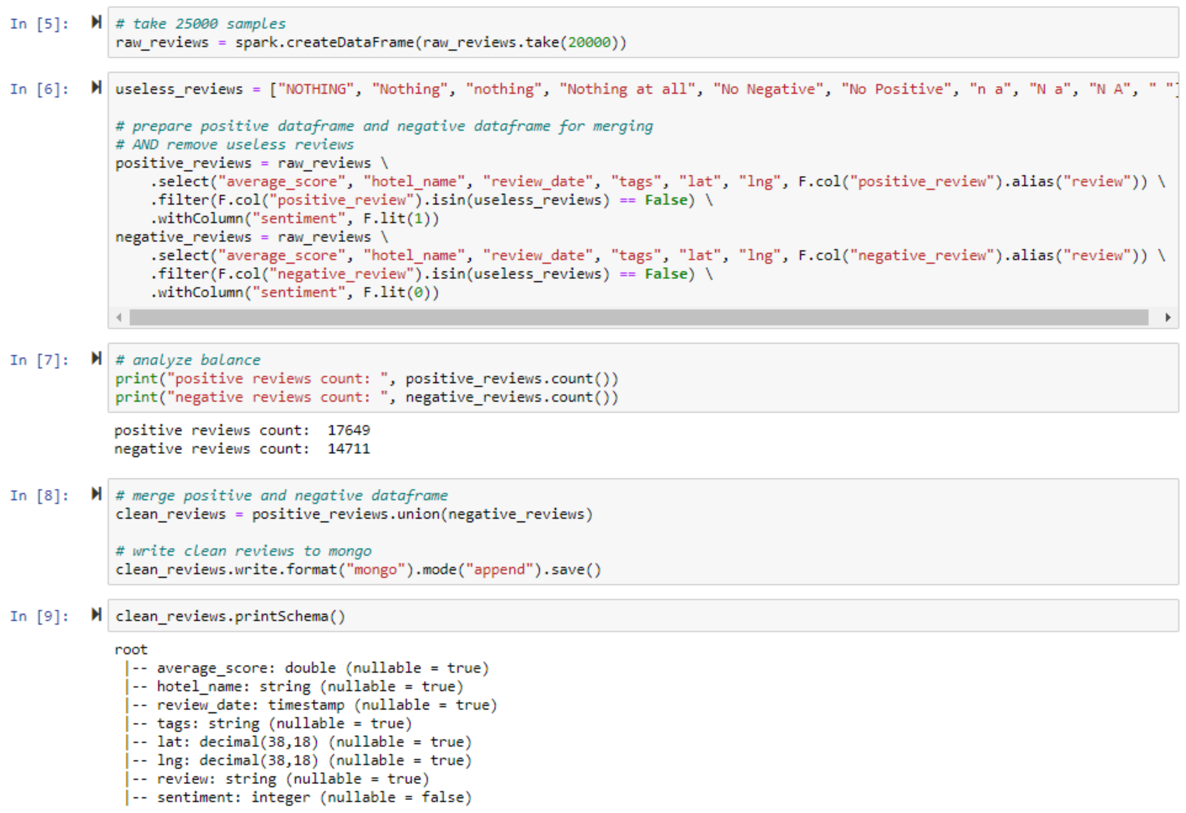


```
In [10]:  # write raw reviews to mongo
          raw_reviews.write.format("mongo").mode("append").save()
```

Figure 7. Writing raw reviews into MongoDB using Spark.

3.1.2 Data preparation

After the raw data is in MongoDB, we create a new session in Spark for getting the raw reviews and cleaning them such that it can be used for the dashboard, as well as for the ML algorithms.



```
In [5]:  # take 25000 samples
         raw_reviews = spark.createDataFrame(raw_reviews.take(20000))

In [6]:  useless_reviews = ["NOTHING", "Nothing", "nothing", "Nothing at all", "No Negative", "No Positive", "n a", "N a", "N A", " "]

         # prepare positive dataframe and negative dataframe for merging
         # AND remove useless reviews
         positive_reviews = raw_reviews \
             .select("average_score", "hotel_name", "review_date", "tags", "lat", "lng", F.col("positive_review").alias("review")) \
             .filter(F.col("positive_review").isin(useless_reviews) == False) \
             .withColumn("sentiment", F.lit(1))
         negative_reviews = raw_reviews \
             .select("average_score", "hotel_name", "review_date", "tags", "lat", "lng", F.col("negative_review").alias("review")) \
             .filter(F.col("negative_review").isin(useless_reviews) == False) \
             .withColumn("sentiment", F.lit(0))

In [7]:  # analyze balance
         print("positive reviews count: ", positive_reviews.count())
         print("negative reviews count: ", negative_reviews.count())

         positive reviews count:  17649
         negative reviews count:  14711

In [8]:  # merge positive and negative dataframe
         clean_reviews = positive_reviews.union(negative_reviews)

         # write clean reviews to mongo
         clean_reviews.write.format("mongo").mode("append").save()

In [9]:  clean_reviews.printSchema()

         root
          |-- average_score: double (nullable = true)
          |-- hotel_name: string (nullable = true)
          |-- review_date: timestamp (nullable = true)
          |-- tags: string (nullable = true)
          |-- lat: decimal(38,18) (nullable = true)
          |-- lng: decimal(38,18) (nullable = true)
          |-- review: string (nullable = true)
          |-- sentiment: integer (nullable = false)
```

Figure 8. Cleaning the dataframe in Spark to be used by the ML algorithms as well as the dashboard.

After this, we insert the data into MongoDB once again, but this time in a different collection: /clean.

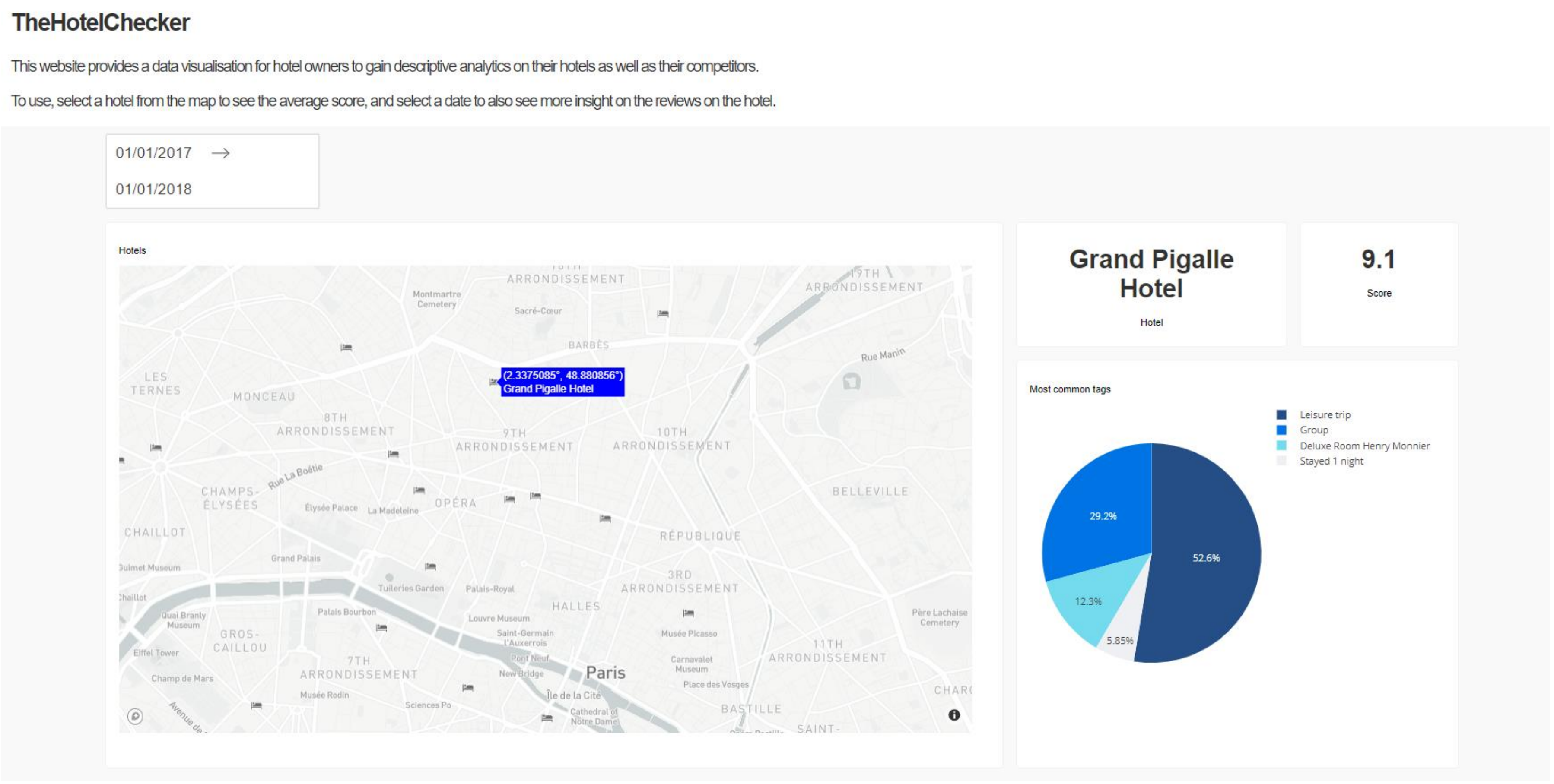
3.1.3 Model building

See 3.2 for model building in-depth.

3.1.4 Data visualization

As mentioned before, the application we're building is for hotel owners to gain insights on their own hotels and their competitors concerning ratings, reviews, tags given by visitors. The dashboard is interactive, giving the user the ability to select a hotel from the interactive map and see the reviews in the requested range of time, as can be seen in Figure x, and x.

It fulfills Schneiderman's mantra's, which is overview (the interactive map), zoom-in-out (date range, map), filter (date range, map), and details-on-demand (selecting a hotel from a map gives details of the hotel).



TheHotelChecker

This website provides a data visualisation for hotel owners to gain descriptive analytics on their hotels as well as their competitors.

To use, select a hotel from the map to see the average score, and select a date to also see more insight on the reviews on the hotel.

01/01/2017 →

01/01/2018

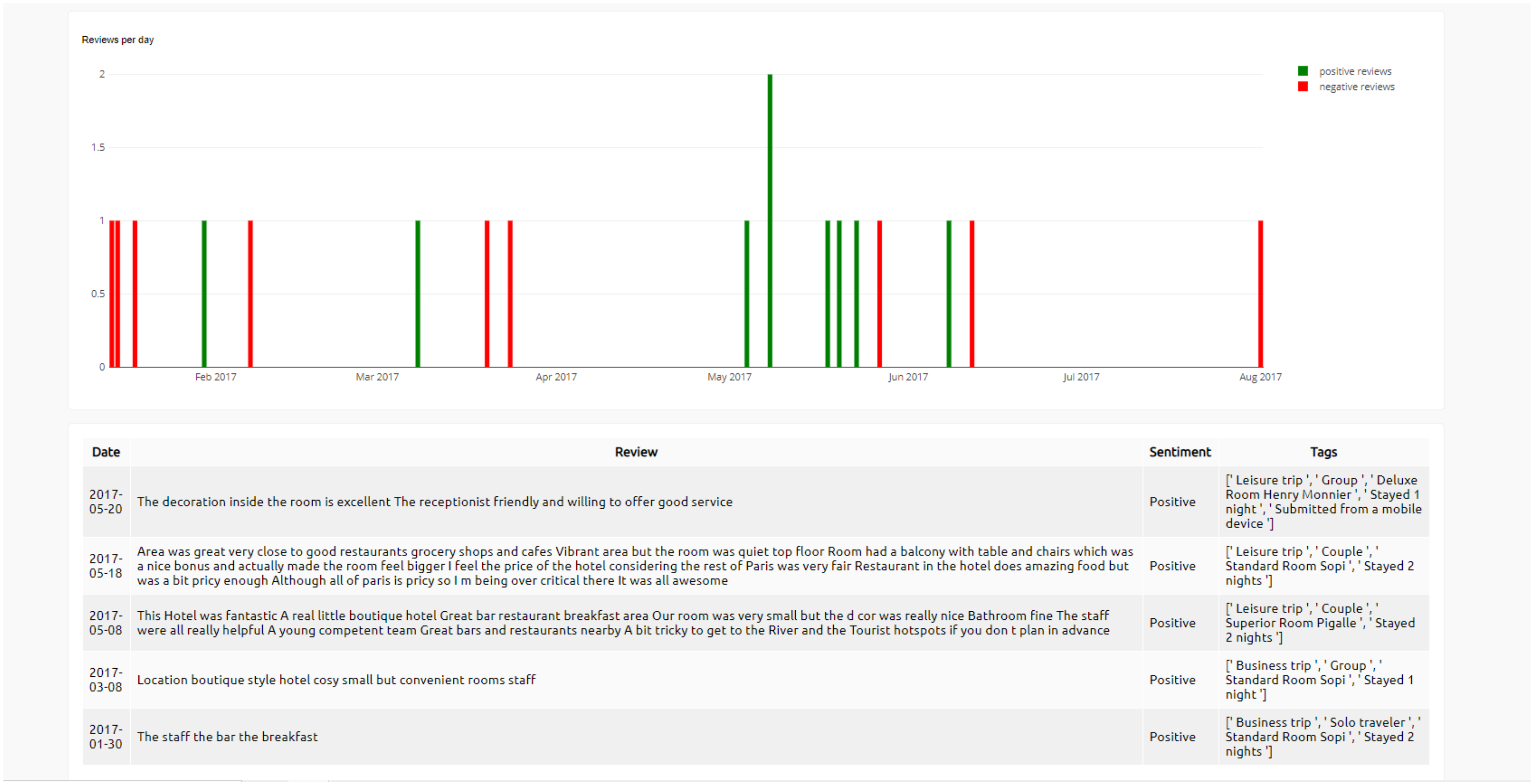
Grand Pigalle Hotel

Hotel

9.1

Score

Figure 9. Upper part of interactive dashboard for hotel owners. Selected Grand Pigalle Hotel, showing its average score and most common tags.



Date	Review	Sentiment	Tags
2017-05-20	The decoration inside the room is excellent The receptionist friendly and willing to offer good service	Positive	[' Leisure trip ', ' Group ', ' Deluxe Room Henry Monnier ', ' Stayed 1 night ', ' Submitted from a mobile device ']
2017-05-18	Area was great very close to good restaurants grocery shops and cafes Vibrant area but the room was quiet top floor Room had a balcony with table and chairs which was a nice bonus and actually made the room feel bigger I feel the price of the hotel considering the rest of Paris was very fair Restaurant in the hotel does amazing food but was a bit pricy enough Although all of paris is pricy so I m being over critical there It was all awesome	Positive	[' Leisure trip ', ' Couple ', ' Standard Room Sopi ', ' Stayed 2 nights ']
2017-05-08	This Hotel was fantastic A real little boutique hotel Great bar restaurant breakfast area Our room was very small but the d cor was really nice Bathroom fine The staff were all really helpful A young competent team Great bars and restaurants nearby A bit tricky to get to the River and the Tourist hotspots if you don t plan in advance	Positive	[' Leisure trip ', ' Couple ', ' Superior Room Pigalle ', ' Stayed 2 nights ']
2017-03-08	Location boutique style hotel cosy small but convenient rooms staff	Positive	[' Business trip ', ' Group ', ' Standard Room Sopi ', ' Stayed 1 night ']
2017-01-30	The staff the bar the breakfast	Positive	[' Business trip ', ' Solo traveler ', ' Standard Room Sopi ', ' Stayed 2 nights ']

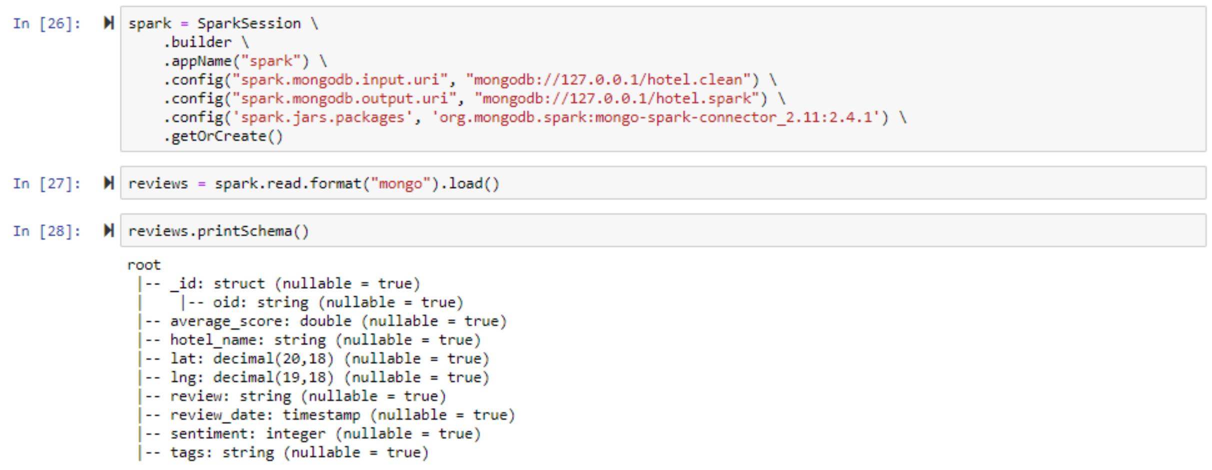
Figure 10. Lower part of interactive dashboard for hotel owners. Selected Grand Pigalle Hotel, showing its reviews and details, sentiment, timeline of reviews.

3.2 In-depth: Model Building

First, we'll take a look at logistic regression in Spark-MLlib, followed by a NN in TensorFlow.

3.2.1 Logistic Regression in Spark-MLlib

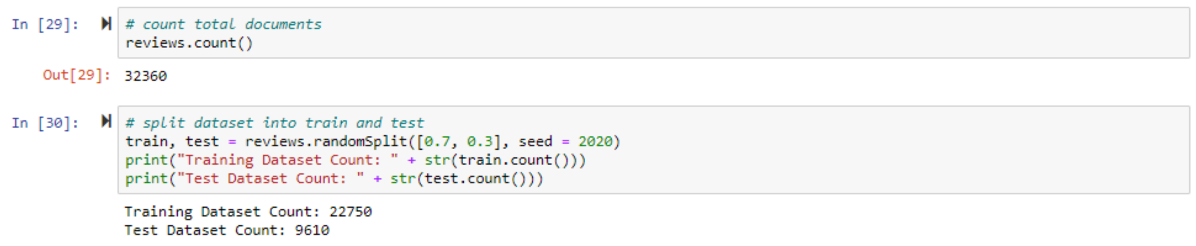
As always, we first read in the data (cleaned from the data preparation phase).



```
In [26]:  spark = SparkSession \
              .builder \
              .appName("spark") \
              .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/hotel.clean") \
              .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/hotel.spark") \
              .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.11:2.4.1') \
              .getOrCreate()

In [27]:  reviews = spark.read.format("mongo").load()

In [28]:  reviews.printSchema()

          root
           |-- _id: struct (nullable = true)
           |    |-- oid: string (nullable = true)
           |-- average_score: double (nullable = true)
           |-- hotel_name: string (nullable = true)
           |-- lat: decimal(20,18) (nullable = true)
           |-- lng: decimal(19,18) (nullable = true)
           |-- review: string (nullable = true)
           |-- review_date: timestamp (nullable = true)
           |-- sentiment: integer (nullable = true)
           |-- tags: string (nullable = true)
```

Figure 11. Reading in data from /clean.

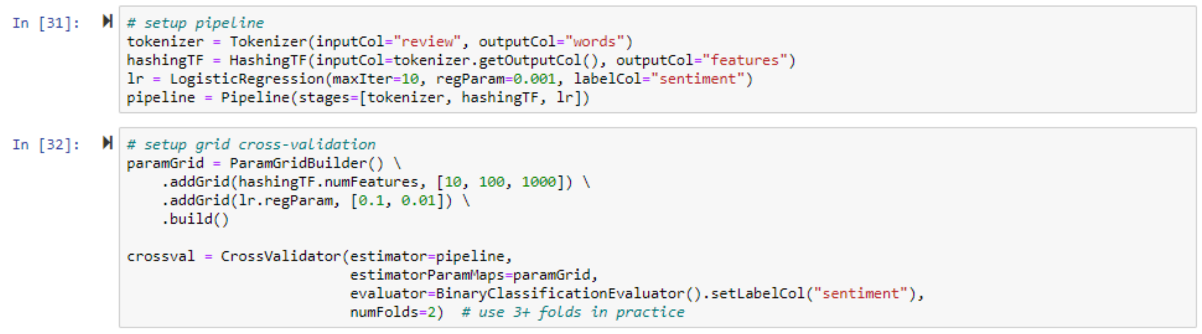
We then shortly analyze our dataset and split it into a train and test dataset.



```
In [29]:  # count total documents
          reviews.count()

Out[29]:  32360

In [30]:  # split dataset into train and test
          train, test = reviews.randomSplit([0.7, 0.3], seed = 2020)
          print("Training Dataset Count: " + str(train.count()))
          print("Test Dataset Count: " + str(test.count()))

          Training Dataset Count: 22750
          Test Dataset Count: 9610
```

Figure 12. Analyzing dataset and splitting.

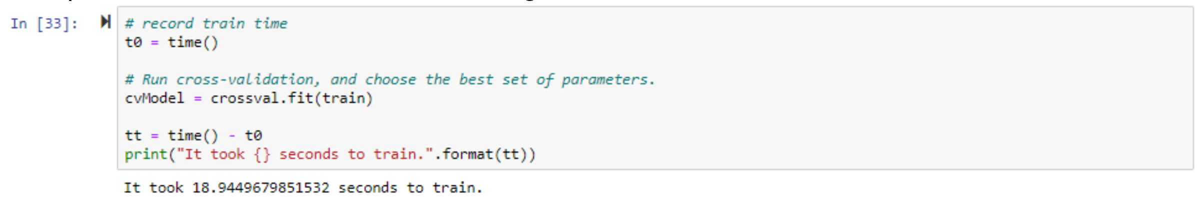
After this we setup a pipeline for tokenizing, hashing, and finally applying logistic regression. We then setup grid cross validation as well with previously made pipeline for tuning hyperparameters of the model.



```
In [31]:  # setup pipeline
          tokenizer = Tokenizer(inputCol="review", outputCol="words")
          hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
          lr = LogisticRegression(maxIter=10, regParam=0.001, labelCol="sentiment")
          pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

In [32]:  # setup grid cross-validation
          paramGrid = ParamGridBuilder() \
              .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \
              .addGrid(lr.regParam, [0.1, 0.01]) \
              .build()

          crossval = CrossValidator(estimator=pipeline,
                                    estimatorParamMaps=paramGrid,
                                    evaluator=BinaryClassificationEvaluator().setLabelCol("sentiment"),
                                    numFolds=2)  # use 3+ folds in practice
```

Figure 13. Setting up pipeline for tokenizing, hashing, and logistic regression. Using grid cross-validation for tuning hyperparameters.

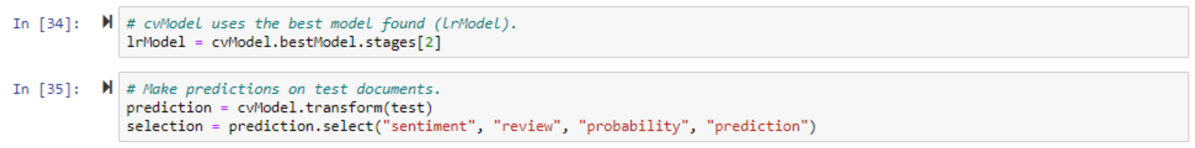
Subsequently, we get to the part of training as can be seen in Figure 14. We add time to see how long the computation will take for later benchmarking.



```
In [33]:  # record train time
          t0 = time()

          # Run cross-validation, and choose the best set of parameters.
          cvModel = crossval.fit(train)

          tt = time() - t0
          print("It took {} seconds to train.".format(tt))

          It took 18.9449679851532 seconds to train.
```

Figure 14. Training the model with grid cross validation and recording the time it takes to train.

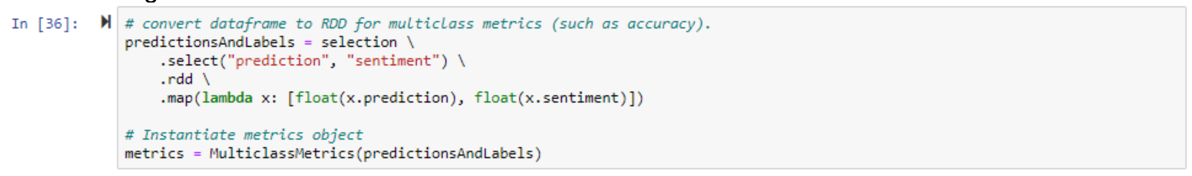
We then use the best estimator to predict the test set.



```
In [34]:  # cvModel uses the best model found (lrModel).
          lrModel = cvModel.bestModel.stages[2]

In [35]:  # Make predictions on test documents.
          prediction = cvModel.transform(test)
          selection = prediction.select("sentiment", "review", "probability", "prediction")
```

Figure 15. Applying best estimator to test set

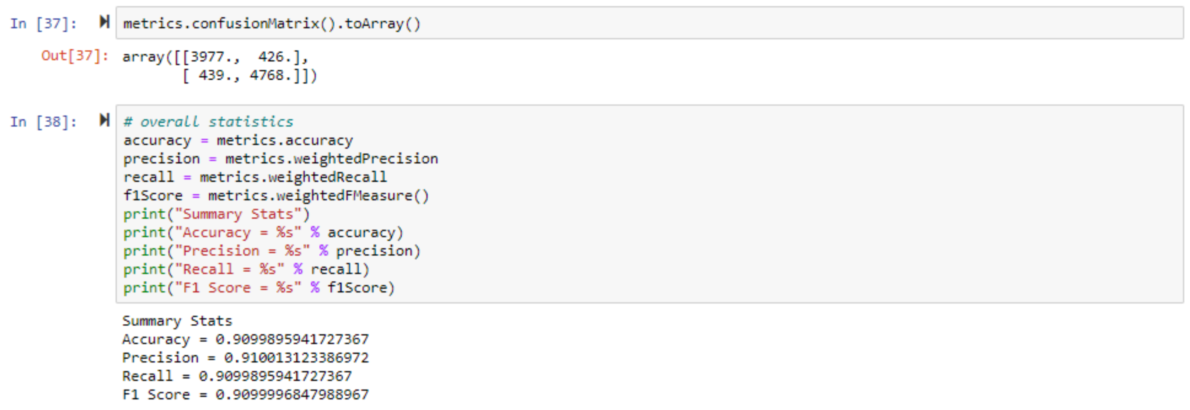
Unfortunately, Spark has inconsistencies between the ML library for RDDs and for Spark DataFrames. The Spark DataFrames library doesn't have simple metrics such as accuracy, but the library for RDDs does. Since, we're using the DataFrames library we have to convert the DataFrame to its equivalent RDD and then get the metrics.



```
In [36]:  # convert dataframe to RDD for multiclass metrics (such as accuracy).
          predictionsAndLabels = selection \
              .select("prediction", "sentiment") \
              .rdd \
              .map(lambda x: [float(x.prediction), float(x.sentiment)])

          # Instantiate metrics object
          metrics = MulticlassMetrics(predictionsAndLabels)
```

Figure 16. Converting DataFrame to RDD for more metrics.

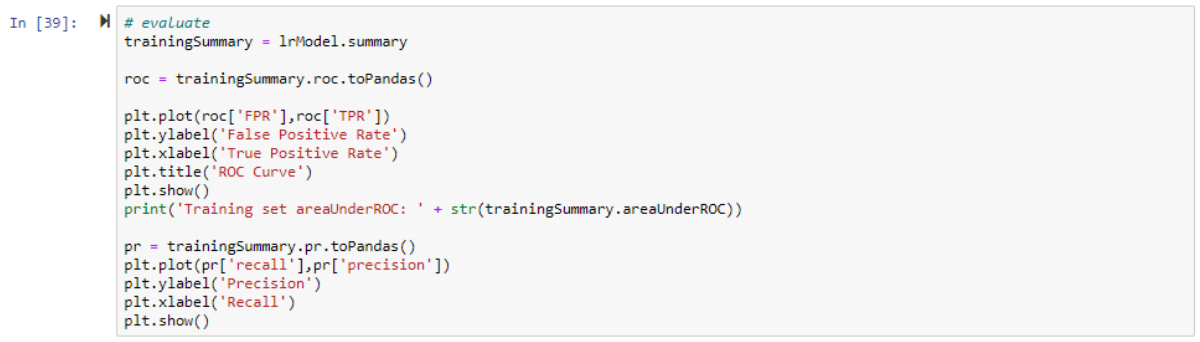
Now, we can finally evaluate our model by getting as many of its basic metrics.



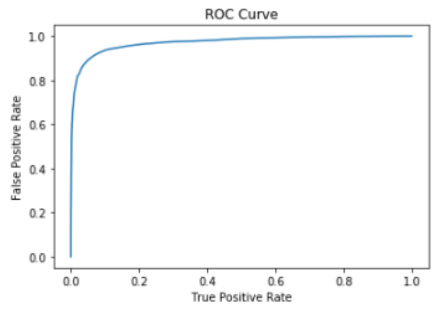
```
In [37]:  metrics.confusionMatrix().toArray()

Out[37]:  array([[3977.,  426.],
                 [ 439., 4768.]])

In [38]:  # overall statistics
          accuracy = metrics.accuracy
          precision = metrics.weightedPrecision
          recall = metrics.weightedRecall
          f1Score = metrics.weightedFMeasure()
          print("Summary Stats")
          print("Accuracy = %s" % accuracy)
          print("Precision = %s" % precision)
          print("Recall = %s" % recall)
          print("F1 Score = %s" % f1Score)

          Summary Stats
          Accuracy = 0.9099895941727367
          Precision = 0.910013123386972
          Recall = 0.9099895941727367
          F1 Score = 0.9099996847988967
```

Figure 17. Evaluate model on basic metrics.

To add upon this, we can dig a bit deeper by looking at the AUC - ROC.



```
In [39]:  # evaluate
          trainingSummary = lrModel.summary

          roc = trainingSummary.roc.toPandas()

          plt.plot(roc['FPR'],roc['TPR'])
          plt.ylabel('False Positive Rate')
          plt.xlabel('True Positive Rate')
          plt.title('ROC Curve')
          plt.show()
          print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))

          pr = trainingSummary.pr.toPandas()
          plt.plot(pr['recall'],pr['precision'])
          plt.ylabel('Precision')
          plt.xlabel('Recall')
          plt.show()
```



```
Training set areaUnderROC: 0.9718380401867147
```

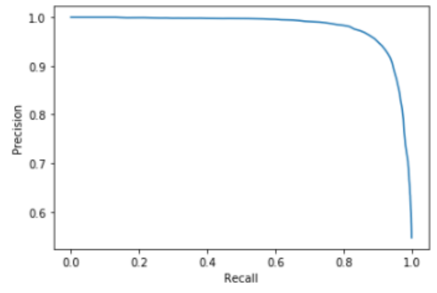
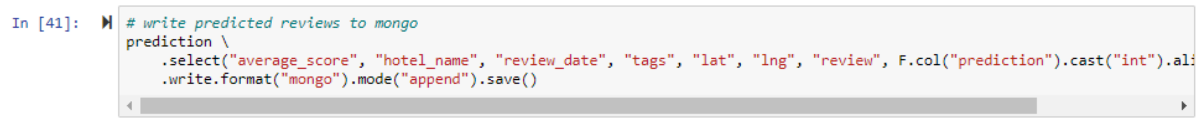


Figure 18. AUC – ROC of logistic regression model.

Finally, we write the predicted dataset to our /spark collection.

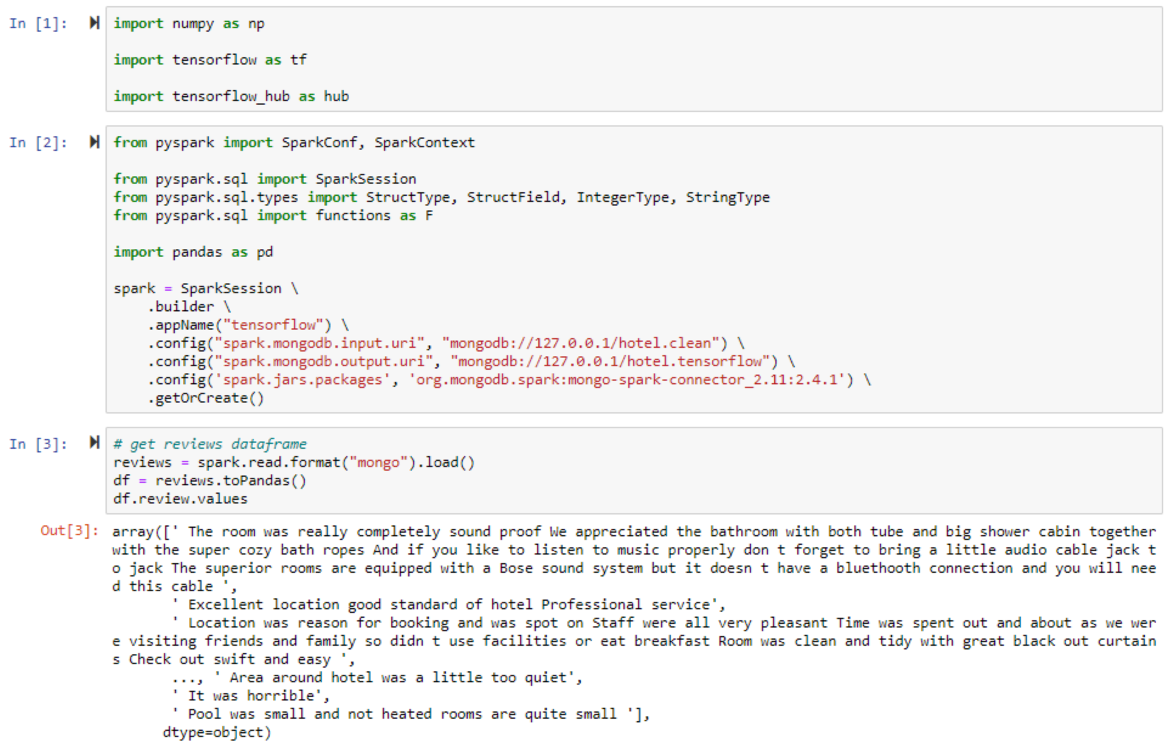


```
In [41]:  # write predicted reviews to mongo
          prediction \
              .select("average_score", "hotel_name", "review_date", "tags", "lat", "lng", "review", F.col("prediction").cast("int").ali
              .write.format("mongo").mode("append").save()
```

Figure 19. Writing predicted dataset to MongoDB: /spark collection.

3.2.2 Neural Network in TensorFlow

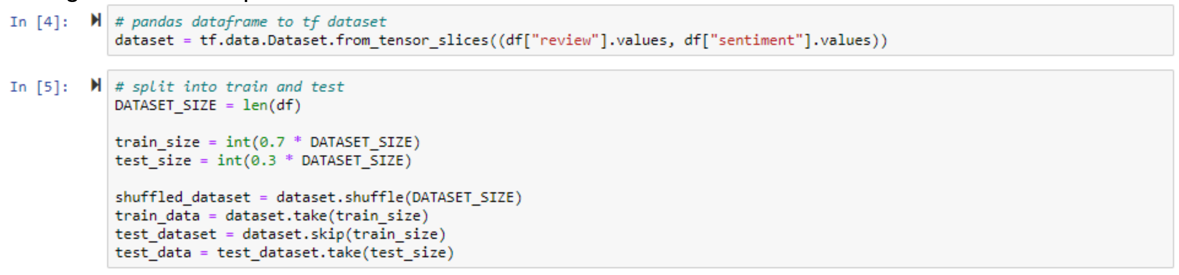
Just as in Spark-MLlib we import our dependencies and create a new Spark session. After getting the RDD, however, we convert this to a Pandas DataFrame, since it will be easier to work with this instead of a Spark DataFrame or RDD for the next step.



```
In [1]:  import numpy as np

         import tensorflow as tf

         import tensorflow_hub as hub

In [2]:  from pyspark import SparkConf, SparkContext

         from pyspark.sql import SparkSession
         from pyspark.sql.types import StructType, StructField, IntegerType, StringType
         from pyspark.sql import functions as F

         import pandas as pd

         spark = SparkSession \
             .builder \
             .appName("tensorflow") \
             .config("spark.mongodb.input.uri", "mongodb://127.0.0.1/hotel.clean") \
             .config("spark.mongodb.output.uri", "mongodb://127.0.0.1/hotel.tensorflow") \
             .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.11:2.4.1') \
             .getOrCreate()

In [3]:  # get reviews dataframe
         reviews = spark.read.format("mongo").load()
         df = reviews.toPandas()
         df.review.values

Out[3]:  array([' The room was really completely sound proof We appreciated the bathroom with both tube and big shower cabin together
         with the super cozy bath ropes And if you like to listen to music properly don t forget to bring a little audio cable jack t
         o jack The superior rooms are equipped with a Bose sound system but it doesn t have a bluethooth connection and you will nee
         d this cable ',
                ' Excellent location good standard of hotel Professional service',
                ' Location was reason for booking and was spot on Staff were all very pleasant Time was spent out and about as we wer
         e visiting friends and family so didn t use facilities or eat breakfast Room was clean and tidy with great black out curtain
         s Check out swift and easy ',
                ..., ' Area around hotel was a little too quiet',
                ' It was horrible',
                ' Pool was small and not heated rooms are quite small '],
               dtype=object)
```

Figure 20. Import dependencies, create Spark session, and convert reviews RDD to Pandas DataFrame.

Now, what we will be doing is converting this Pandas DataFrame containing all of our reviews and sentiment to a Tensor. After that, we shuffle and split the dataset in the same distribution as we did in logistic regression and Spark-MLlib.

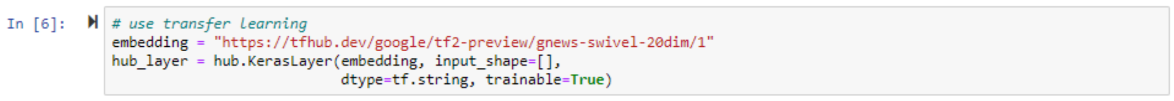


```
In [4]:  # pandas dataframe to tf dataset
         dataset = tf.data.Dataset.from_tensor_slices((df["review"].values, df["sentiment"].values))

In [5]:  # split into train and test
         DATASET_SIZE = len(df)

         train_size = int(0.7 * DATASET_SIZE)
         test_size = int(0.3 * DATASET_SIZE)

         shuffled_dataset = dataset.shuffle(DATASET_SIZE)
         train_data = dataset.take(train_size)
         test_dataset = dataset.skip(train_size)
         test_data = test_dataset.take(test_size)
```

Figure 21. Creating Tensor and splitting dataset.

Now, we get to building the model. For the first layer, I used a pre-trained text embedding model from TensorFlow Hub. I could have also converted the text to embeddings vectors myself, but using a pre-trained model has several advantages, such as:

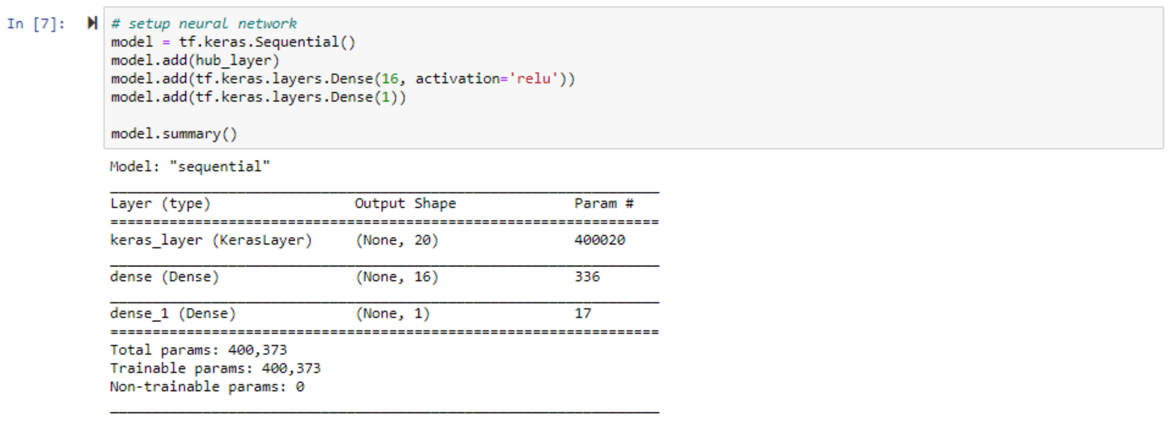
- No text preprocessing needed.
- Benefit from transfer learning, which means using a pre-trained model that was actually used for another specific purpose, but also is useful for this task.
- The embedding has a fixed size, so it’s simpler to process (*TensorFlow Hub*, 2020).



```
In [6]:  # use transfer learning
         embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"
         hub_layer = hub.KerasLayer(embedding, input_shape=[],
                                    dtype=tf.string, trainable=True)
```

Figure 22. Creating transfer learning layer.

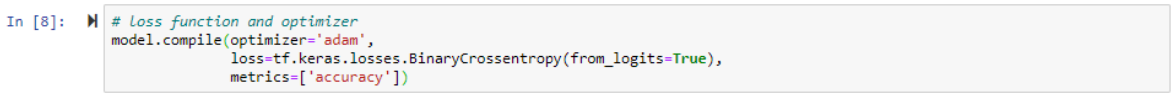
Subsequently, we design our neural network with the help of the previous layer. What's happening here is that we set the embedding layer as our first layer. We have to, since we cannot do ML on strings of text, but only its vectors. Then, we add on this a Dense layer which ReLU as its activation function, which sets negative values to zero and positive values to just their positive values. The motivation for the size of this layer is none, since only by experiment we can find the best size for layers. And finally, we just have our output layer, which is just one node. The reason for this is because we perform binary classification. In the case of multiclassification, we'd have more nodes there.



```
In [7]:  # setup neural network
         model = tf.keras.Sequential()
         model.add(hub_layer)
         model.add(tf.keras.layers.Dense(16, activation='relu'))
         model.add(tf.keras.layers.Dense(1))

         model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         keras_layer (KerasLayer)     (None, 20)                400020
         _________________________________________________________________
         dense (Dense)                (None, 16)                336
         _________________________________________________________________
         dense_1 (Dense)              (None, 1)                 17
         =================================================================
         Total params: 400,373
         Trainable params: 400,373
         Non-trainable params: 0
         _________________________________________________________________
```

Figure 23. Designing our neural network.

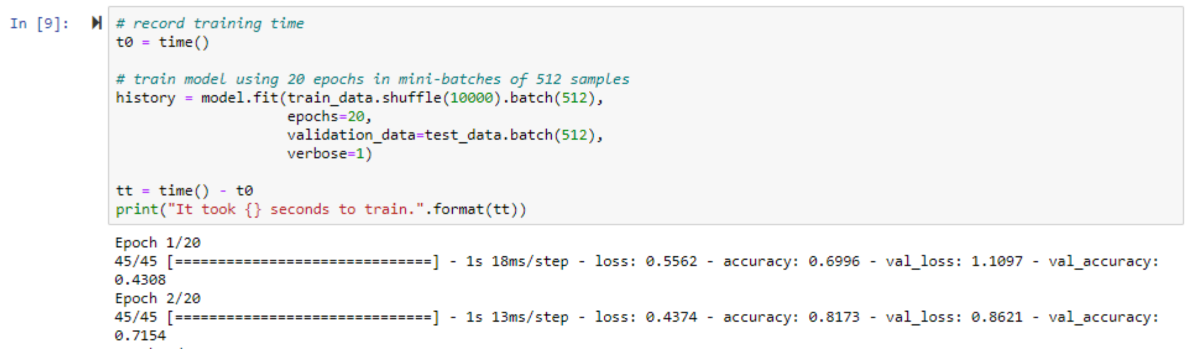
Next, we setup our loss function and optimizer. This is necessary for back propagation, for the neural network to basically learn. We pick for our loss function binary cross entropy, since we're dealing with a binary classification problem. Other options are also available, but for simplicity, we use this. As optimizer, we use 'adam', which is a stochastic gradient descent method. The goal of the optimizer is to update the weights to minimize the loss function (Renu Khandelwal, 2019).



```
In [8]:  # loss function and optimizer
         model.compile(optimizer='adam',
                       loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                       metrics=['accuracy'])
```

Figure 24. Setup a loss function and optimizer.

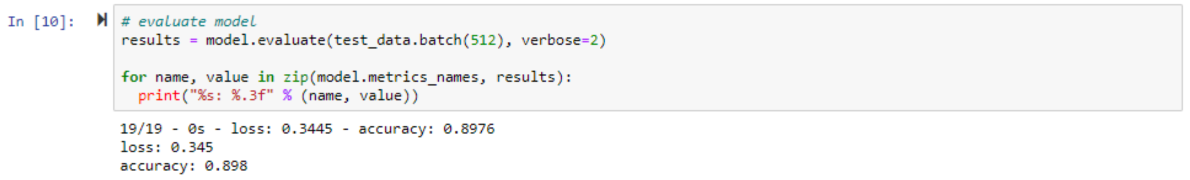
We now finally get to training our neural network. This is done in 20 epochs (a.k.a. dataset passed forward and backward through our neural network) of a batch of 512 samples of our training data.



```
In [9]:  # record training time
         t0 = time()

         # train model using 20 epochs in mini-batches of 512 samples
         history = model.fit(train_data.shuffle(10000).batch(512),
                             epochs=20,
                             validation_data=test_data.batch(512),
                             verbose=1)

         tt = time() - t0
         print("It took {} seconds to train.".format(tt))

         Epoch 1/20
         45/45 [==============================] - 1s 18ms/step - loss: 0.5562 - accuracy: 0.6996 - val_loss: 1.1097 - val_accuracy:
         0.4308
         Epoch 2/20
         45/45 [==============================] - 1s 13ms/step - loss: 0.4374 - accuracy: 0.8173 - val_loss: 0.8621 - val_accuracy:
         0.7154
```

Figure 25. Training our neural network.

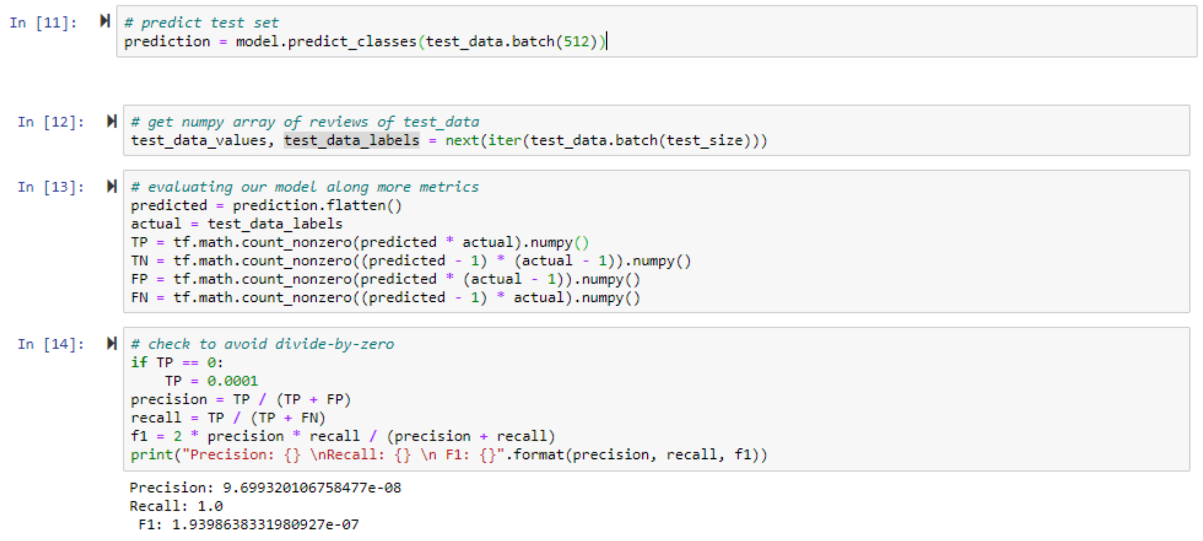
Next, we evaluate our model using our test set. As we can see, we were able to get an accuracy of almost 90%.



```
In [10]:  # evaluate model
          results = model.evaluate(test_data.batch(512), verbose=2)

          for name, value in zip(model.metrics_names, results):
            print("%s: %.3f" % (name, value))

          19/19 - 0s - loss: 0.3445 - accuracy: 0.8976
          loss: 0.345
          accuracy: 0.898
```

Figure 26. Evaluate neural network using test dataset.

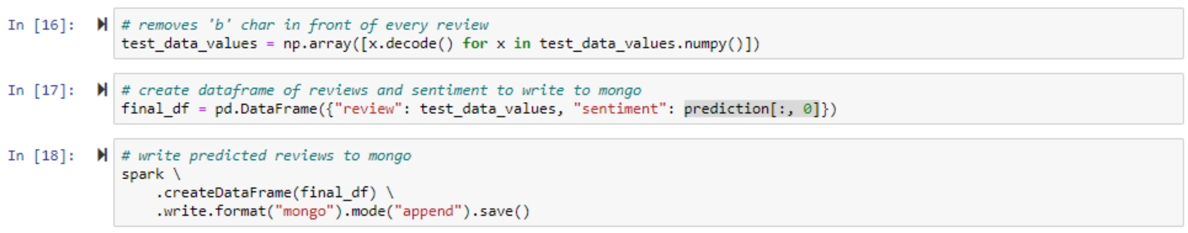
After this, we close off by predicting our test set and gather more metrics.



```
In [11]:  # predict test set
          prediction = model.predict_classes(test_data.batch(512))


In [12]:  # get numpy array of reviews of test_data
          test_data_values, test_data_labels = next(iter(test_data.batch(test_size)))

In [13]:  # evaluating our model along more metrics
          predicted = prediction.flatten()
          actual = test_data_labels
          TP = tf.math.count_nonzero(predicted * actual).numpy()
          TN = tf.math.count_nonzero((predicted - 1) * (actual - 1)).numpy()
          FP = tf.math.count_nonzero(predicted * (actual - 1)).numpy()
          FN = tf.math.count_nonzero((predicted - 1) * actual).numpy()

In [14]:  # check to avoid divide-by-zero
          if TP == 0:
              TP = 0.0001
          precision = TP / (TP + FP)
          recall = TP / (TP + FN)
          f1 = 2 * precision * recall / (precision + recall)
          print("Precision: {} \nRecall: {} \n F1: {}".format(precision, recall, f1))

          Precision: 9.699320106758477e-08
          Recall: 1.0
           F1: 1.9398638331980927e-07
```

Figure 27. Predicting test dataset and sending this to a separate MongoDB collection.

Finally, we send this off to MongoDB. Also, we have to perform a bit of cleaning and merging to send a correct dataset to MongoDB. We know our predictions, but we also need the associated reviews.



```
In [16]:  # removes 'b' char in front of every review
          test_data_values = np.array([x.decode() for x in test_data_values.numpy()])

In [17]:  # create dataframe of reviews and sentiment to write to mongo
          final_df = pd.DataFrame({"review": test_data_values, "sentiment": prediction[:, 0]})

In [18]:  # write predicted reviews to mongo
          spark \
              .createDataFrame(final_df) \
              .write.format("mongo").mode("append").save()
```

Figure 28. Cleaning, merging, and sending data off to MongoDB.

4. Benchmarks

Now we have seen how the models have been implemented in our case study. After running the application we will now look at the benchmark results of the classifiers.

Train Time

Logistic Regression	Neural Network
18.94	**14.29**

Accuracy

Logistic Regression	Neural Network
0.909	**0.898**

Precision

Logistic Regression	Neural Network
0.910	**0** (due to low True Positives)

Recall

Logistic Regression	Neural Network
0.909	**0** (due to low True Positives)

F1

Logistic Regression	Neural Network
0.909	**0** (due to low True Positives)

Confusion Matrix of Logistic Regression

	Actual Positive	Actual Negative
Predicted Positive	**3977**	**426**
Predicted Negative	**439**	**4768**

Confusion Matrix of Neural Network

	Actual Positive	Actual Negative
Predicted Positive	**0**	**1031**
Predicted Negative	**0**	**8677**

As we can see, our neural network performed under the expectations. Zero true positives and false negatives leads to nonsense precision, recall, and f1 scores. On the other hand, we do see that our neural network was trained in a short time, even shorter than logistic regression. Also, we haven't trained our classifiers with all the 515k reviews, but rather a small subset of ~20k. Neural networks are notorious for needing large datasets to be of good use. Having said this, we do see that logistic regression is a relatively good ML model for the dataset size given and beats the neural network.

5. Conclusion

Let’s return to our hypothesis, which was the following:

> *Neural networks (built with TensorFlow) perform better than logistic regression models (built with Spark-MLlib) in terms of basic metrics (accuracy, time to train, precision, recall, f1-score).*

Rather, obviously our neural network performed under the expectations. Most probably, due to the small dataset used for training, since neural networks are notorious for needing large datasets.

Anyhow, the logistic regression from Spark-MLlib did outperform the neural network built in Tensorflow given this small dataset. For future research, it would be interesting to increase the size of the dataset and see what results we get.

Therefore, neural networks (built with TensorFlow) do not necessarily perform better than logistic regression (built with Spark-MLlib) in terms of basic metrics.

The dependent factors to this are unclear but might likely have to do with the size of the dataset.

Also, interesting is that the logistic regression model in Spark-MLlib actually took longer than the neural network in TensorFlow to be trained. Expected was that the training time of Spark-MLlib would be faster considering that it processes datasets in parallel.

Bibliography

TensorFlow. (2020). TensorFlow. https://www.tensorflow.org/

MLlib | Apache Spark. (2020). Apache.Org. https://spark.apache.org/mllib/

sklearn.linear_model.LogisticRegression — scikit-learn 0.23.1 documentation. (2014). Scikit-Learn.Org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logistic%20regression#sklearn.linear_model.LogisticRegression

Spark 101: What Is It, What It Does, and Why It Matters | MapR. (2013). Mapr.Com. https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters/

Villu Ruusmann. (2017, March 22). *R, Scikit-Learn and Apache Spark ML - What difference does it make?* Slideshare.Net. https://www.slideshare.net/VilluRuusmann/r-scikitlearn-and-apache-spark-ml-what-difference-does-it-make

What is end to end learning in machine learning? - Quora. (2019). Quora.Com. https://www.quora.com/What-is-end-to-end-learning-in-machine-learning

3Blue1Brown. (2020). But what is a Neural Network? | Deep learning, chapter 1 [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=aircAruvnKk&t=731s

TensorFlow Hub. (2020). TensorFlow. https://www.tensorflow.org/hub