Spark on EMR Cluster

Alaleh Hamidi, Masoumeh Nourollahi, Soude Ghari

Abstract

Data volumes has been growing increasingly in industry and research , which poses tremendous opportunities, as well as great computational challenges. Since data sizes have outpaced the capabilities of single machines, users have required innovative systems to scale out computations to multiple nodes. Consequently, there has been an explosion of new cluster programming models aiming varied computing workloads. At first, these models were relatively specialized, with new models developed for new workloads. In this paper, we write a simple wordcount code using spark on an EC2 instance. After that, we analysed 3 machine learning models on AWS EMR cluster.

1 Introduction

A new model of cluster computing has gained extensively popularity, in which data-parallel computations are executed on clusters of unreliable machines by systems that automatically provide locality-aware scheduling, fault tolerance, and load balancing which is well-known as Apache Spark. Apache Spark is an open-source, general-purpose distributed computing system used for big data analytics. In comparison with Apache Hadoop, Spark is able to complete jobs substantially faster because of its in-memory caching, and optimized query execution. Spark provides development APIs in Python, Java, Scala and other language.

In this paper, we use MapReduce on Spark using AWS while reading data from S3 which stands for Amazon Simple Storage service and deploy an ML algorithm into AWS EMR cluster. Firstly, it had to be installed Spark with all its requirements (python3.7, awscli, python3-pip, defaultjre, default-jdk) and configure PySpark, so that we can run it on the instance t2.2xlarge. Then, we used Amazon S3 to store and retrieve data in order to launch a script for counting words.

Section 2 presents the methodology used to setting up Spark and S3 bucket. Section 3 propose experiment with WordCount and section 4 presents input data analysis for the ML models. Section 5 introduces building machine learning on AWS EMR Cluster . Finally section 6 presents the result accuracy.

2 Word Count Program

2.1 Setting up Spark and S3 bucket

Apache Spark is a batch and stream processing engine. It has been proven to be almost 100 times faster than Apache Hadoop and much easier to develop distributed big data applications with. We applied 2 different approach to install Spark on EC2 instance. We explain both of them. We applied PIP for installing Spark on targeted instance(t2.2xlarge). We also explained how to setup saprk on

EMR cluster in the next section. After providing all required packages, we installed pyspark with this command:

```
pip install pyspark
```

For the second approach we followed the next steps:

1. Installing the per-requisites of Spark

   - Installing open-jdk. A note here is that first we installed latest version on open-jdk which was version 11. Later we had issues with running our code in spark and understood that spark is compatible with open-jdk 8. Also after installing open-jdk package the related environment variables should be set in bashrc. Following command was used to first update the ubuntu repositories and then install open-jdk-8

```
sudo apt-get update
sudo apt-get upgrade
sudo apt install openjdk-8-jdk
```

   - Installing python3. Also same as java related environment variable for python should be set in bashrc file

```
sudo apt-get install python3.7
```

   - Installing pip

```
sudo apt-get install python3-pip
```

   - Installing boto3. This tool is used for downloading and uploading to S3 bucket

```
pip install boto3
```

2. Downloading and extracting spark. Also spark-home and path should be defined in this step.

```
sudo mkdir /usr/local/spark
cd /usr/local/spark/
sudo wget http://apache.forsale.plus/spark/spark-2.4.5/spark-2.4.5-bin-hadoop2.7.tgz
sudo tar xvzf spark-2.4.5-bin-hadoop2.7.tgz
```

3. Installing awscli and its configuration. awscli installation is discussed in detail in 2.2 section.
4. Following are all the environment variables that are defined for java, python and spark configuration:

$export\ JAVA_HOME = /usr/lib/jvm/java - 8 - openjdk - amd64$
$exportSPARK_HOME = /usr/local/spark/spark - 2.4.5 - bin - hadoop2.7$
$exportPATH = PATH{:}JAVA_HOME/bin\ {:}SPARK_HOME/bin$
$exportPYTHONPATH = SPARK_HOME/python\ {:}SPARK_HOME/python/lib/py4j -$
$0.10.7 - src.zip : PYTHONPATH$
$exportPYSPARK_PYTHON = python3$

By doing all these configurations system is now ready to test the word count code.

2.2 Creating and uploading a file into S3 bucket

Firstly, we need to create a S3 bucket ,in order to process our text file with Spark. Amazon Simple Storage Service (Amazon S3) is a cost effective storage. We can apply Amazon S3 to store and retrieve any amount of data. After that, we upload the intended file to our S3 bucket, then use Spark to process it. We use AWS CLI library to handle movements of our data. AWS CLI(Command Line Interface) has made working with S3 very easily. The syntax for copying files to/from S3 in AWS CLI is:

```
sudo pip install awscli
aws configure
aws access key:
aws secret access key:
region:
```

All the required data can be obtained from AWS account. Moreover, creating a new bucket is done with the following command:

```
aws s3 mb s3://name-of-bucket --region us-east-1
```

mb stands for make bucket. For copying data to S3 bucket, we followed this step:

```
aws s3 cp (source) (destination)
```

2.3 Result of Word Count Program

The primary abstraction of Spark is that of a resilient distributed dataset(RDD),which presents a read-only collection of objects partitioned across a set of machines that can be rebconstructed if a partition is being lost. The elements of an RDD require not to exist in physical storage; instead, it should handle an RDD comprise adequate information to compute the RDD starting from data in reliable storage. This indicates that RDDs can always be rebuilt if nodes have failure [1].

In this section, with the help of RDD spark, we write a wordcount program that show the frequency of each word of a book file. Word Count is a program, reading text files and then counts how often words happen. The input and output of this program are text files. The program split each line and count how often it occurred, which separated by a tab.

There are two approach for applying operations on these RDD's.

- Transformation
- Action

Transformation are the operations, which can be applied on a RDD to create a new RDD. Action are the operations that can be applied on RDD, which instructs Spark to do computation and sending the result to the driver.

- filter: It is a transformation operation of RDD which accepts some parameter as an argument and returns boolean value. The method will pass these parameter and operates on the source RDD. It will filter all the elements of the source RDD for which the parameter is not satisfied and creates new RDD with the elements. We will explain the special RDD used in writing our word count program.

- Map: Map is considered as the transformation operations in Apach Spark. Map() operation applies to each elements of the RDD and returns the result as new RDD.Spark Map function takes one element as input and returns one element at a time.

- A flatMap is considered as a transformation operation,which applies to each element of RDD. Then, it returns the outcome as new RDD. The functionality of flatMap has similarity with Map, but FlatMap allows returning two value 0, 1 or more elements in comparison with map function.

- ReduceByKey: It is a spark action that aggregates a data set (RDD) element utilising a function.

  By using reduceByKey on a dataset (Key, Value), the pairs on the same machine with the same key are combined, before the data is shuffled.

- sortByKey: By applying this function on a dataset of (Key, Value) pairs, the data is sorted according to the key K in another RDD.

- take(n): The action take(n) returns n number of elements from RDD. It struggles to cut the number of access partition, so it represents a biased collection. We cannot presume the order of the elements. Firstly, the text file "219-0.txt" was stored on the S3. For the implementation of WordCount, we used boto3 to extract the file content. After splitting the text by lines, we parallelized by using a SparkContext. Then, we were able to apply the algorithm given, using mappings and ReduceByKey. The result of WordCount programme is represented as follow: total number of words is: 41456

has 516 counts

would has 130 counts

like has 111 counts

has 107 counts

could has 107 counts

Project has 78 counts

see has 74 counts

man has 71 counts

said has 68 counts

seemed has 66 counts

made has 64 counts

Kurtz has 62 counts

came has 59 counts

without has 57 counts

little has 57 counts

upon has 56 counts

looked has 55 counts

Gutenberg-tm has 53 counts

know has 50 counts

work has 49 counts

3 Analysis of input dataset

The given data set is a description of 12 air pollutant and relevant meteorological variables of 11sites of Beijing. This data set with total 385704 records (35064 records per state) is hourly gathered from 2013-03-01 to 2017-02-28. We got the number of null for each variable using the following code:

```
print(dataset.isnull().sum())
```

All variables except year, monnth, day, hour and station had missing values.Table.1 shows the number of nulls for the variables which contain missing values:

To clean our data set, we decided to replace all missing values except for 'TEMP' variable. As TEMP is the most important variable and is considered as our dependent variable, we decided to remove its missing values to get more accurate results. Since the percentage of removed cases is less than 5% of the observations, we can be confident that removing causes no problem. As we

Table 1: The number of missing values for each variable

PM2.5	PM10	SO2	NO2	CO	O3	TEMP	PRES	DEWP	RAIN	wd	WSPM
8043	5965	8352	11362	19404	12199	379	374	384	371	1743	305

understood 'wd' should be dropped during feature engineering, we didn't apply any cleaning on it. But if we wanted to take it as a feature, we would remove its missing values because of the small number of them. The data set contains two categorical variables (wd and station). 'wd' and 'station' respectively contain 16 unique wind direction and 11 unique sites. The frequency of these values for each variable of 'wd' and 'station' is in the same range. this means that they don't have any outlier. We ran the following code to visualize each variable and understand the probaility distribution of it:

$$seaborn.distplot(dataset['Variable_Name'])$$

As we can see in figure.1, 'RAIN' variable doesn't have any entries for different amounts of 'RAIN'. So, this variable can't be useful for our model and we should drop it.

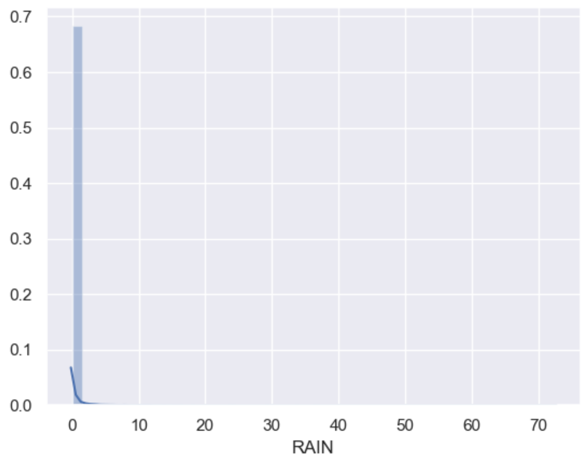

Figure 1: Probability distribution of 'RAIN'

As we are looking for a normal distribution in the feature's plot, we understood that some features like 'CO', 'NO2', 'PM10', and 'WSPM' have some observations which lie on abnormal. Probability distribution of 'CO' represents its outliers which are mostly occurred after the amount of 4000.

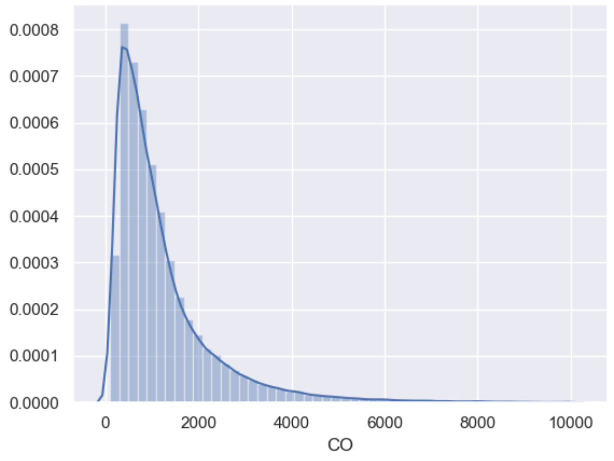

Figure 2: Probability distribution of 'CO'

To deal with these outliers, we removed the top 1% of observations of abnormal features. To build some machine learning models, it is important that variables to be in the same scale. According our data set, all of features except 'PRES', 'year' and 'CO' are almost in the same feature. Since we eliminate these feature during feature engineering, there is no need to apply feature scaling via our python code.

3.1 Feature engineering

We identified the most important features in two phases. In the first phase we obtained the importance of each feature to predict 'TEMP' variable, using Random Forest Regressor. We applied this model on all of variables to understand the effect of them on our dependent variable ('TEMP'). Figure3 shows the result of the first phase of feature engineering.

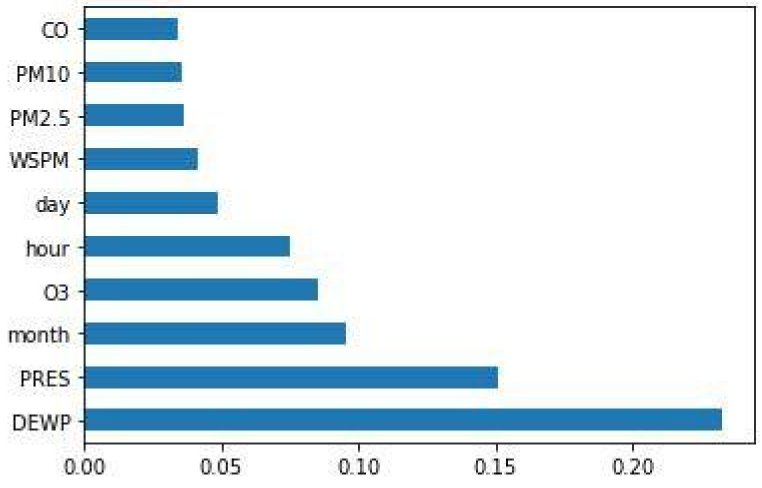

Figure 3: The importance of each feature to predict 'TEMP'

After Understanding the importance of features in the first phase, we applied backward elimination on our models and compared the accuracy of each state. In this phase we could understand the correlation between features. Only one of the features which have correlation should be taken. At last, for both Random Forest and KNN (K-Nearest Neighbors), 'month', 'day', 'hour', 'PM10', 'SO2', 'NO2', 'O3', 'DEWP', 'WSPM' and 'CO' are chosen as important features. All of mentioned features are taken for SVM model (Support Vector Machine) except 'CO'.

4 Description of machine learning models

According to the nature of our data, we should choose supervised algorithms to classify our prediction. We chose SVM, Random Forest and KNN models based on their characteristics.

4.1 Support Vector Machine (SVM)

SVM is a classifier model which creates decision boundaries between data classes. It finds the optimal line/curve in dimensional data space to separate data points to some classes. This line/curve is searched through supporting vectors (Supporting vectors are the very extermely close cases to the boundary). This approach makes the SVM algorithm more special than non-supported of vector machine algorithms. Using the 'kernel' parameter of SVM, we can choose the type of boundary which classifies our data. Since our data is not linearly set separated, we were sure that we shouldn't set kernel parameter as linear. But we chose 'rbc' instead of other values by trial and error. The features we take for our SVM model are shown bellow:

'month', 'day','hour', 'PM10', 'SO2', 'NO2', 'O3', 'DEWP', 'WSPM'

We trained our model by not all of the observation of our data set because of preventing overfeeding. At the end, we obtained 74/48% accuracy for SVM model.

4.2 K-Nearest Neighbors (KNN)

KNN is a supervised algorithm which takes K nearest neighbors of the new data point according to the Euclidean distance. Then, among these K neighbors, it counts the number of data point in each category. At last, KNN assigns the new data point to the category, where most neighbors are counted. As our data is not linearly set separated, we chose this algorithm to classify our data set, because classes don't have to be linearly separable in KNN approach. We set KNN parameters by trying different values for them and after applying greedy search, we understood that the following choices get the best accuracy:

$(n\_neighbors = 5, metric =' minkowski', p = 2)$

The features we chose for our KNN model during the feature engineering, are shown bellow:

'month', 'day','hour', 'PM10', 'SO2', 'NO2', 'O3', 'DEWP', 'WSPM', 'CO'

At the end, we got 71/53% accuracy for our KNN model.

4.3 Random Forest

Random forest classifier is a supervised algorithm which picks at K random data points from the training set and builds the decision trees associated to these K data points. Then it chooses the number Ntree of trees you want to build and again repeats these steps. For a new data pont, Random Forest makes each one of your Ntree trees predict the category to which the data point belongs, and assigns the new data point to the category that wins the majority vote. Because of the predictive performance can compete with the best supervised learning algorithms, we select Random Forest as one of our models. We set Random Forest parameters by trying different values for them and after applying greedy search, we understood that the following choices get the best accuracy:

$$(n_estimators = 100, criterion =' entropy', random_state = 0)$$

The features we chose for our Random Forest model during the feature engineering, are the same as the ones for KNN:

'month', 'day','hour', 'PM10', 'SO2', 'NO2', 'O3', 'DEWP', 'WSPM', 'CO'

At the end, we got 92/52% accuracy for our Random Forest model which was the best accuracy among the two other models.

4.4 Set up and launch EMR cluster

Amazon EMR provides a managed framework that makes it straightforward, fast, and cost-effective to process huge amounts of data across dynamically [2]. Amazon EMR autoscales the cluster and adds or removes nodes once spot instances are turned off/on. Amazon EMR is a completely managed data service based on Apache Spark, which is integrated with the cloud environment of Amazon Web Services (AWS), including its storage service layer known as S3. It is designed to omit the complexity involved in the manual provisioning and setup of data resources, including the Spark cluster, the tuning of the environment. Moreover,the service consists of facilities to make sure that the data is secure, compliant to regulations, and auditable. There are novel approaches to set up and manage machine learning (ML) operations on data in EMR. The advantages of using Apache Spark on Amazon EMR are as follows:

- Quick performance
- Develop applications immediately
- Create a variety of workflows

- Autoscaling cluster

we launch our sample cluster by applying Quick Options in the Amazon EMR console. First, we open the Amazon EMR console and opt ”create cluster”. On the following page, we identify our cluster name and also under the security and access, the created EC2 key pair is chosen. Finally, ”create cluster” button was chosen. It is worth to mention that we wanted to launch our EMR cluster with c5.xlarge instance since according to the previous assignment(TP1),this instance is the best one regarding to 6 applied benchmarking approach, however, we could not do this as not all kinds of instances have access in all regions [3]. Due to this reason, our cluster fails to provision.Therefore, we obliged to use with 1 master node and two work nodes of m4.large instances.

4.5 How to run the program

You can set the parameters in the first part of code, but they have default values and code can be run by them properly. If the 'TEMP column of test data is categorical, you should make the lines 95 to 108 of the code comment and replace y variable with y1 in the lines 121,136 and 152 (The lines in which accuracy is calculated).

References

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica et al., “Spark: Cluster computing with working sets.” HotCloud, vol. 10, no. 10-10, p. 95, 2010.

[2] ”Apache Spark”, Available Online:https://databricks.com/spark/about//.

[3] ”Supported Instance Types”, Available Online: https://docs.aws.amazon.com/emr/latest/ManagementGuid
supported-instance-types.html//.