MapReduce & Machine learning Model in cloud environment

Moses Openja, Mahsa Hadian, Dmytro Humeniuk & Tchanjou Njomou Aquilas

February 2020

Abstract

Big data ecosystem includes multiple applications and frameworks that are often run of different clusters with a cloud environment, to allows to parallel processing as well as reducing the performance overheat. Therefore there is always a need to understand the underlying technology involved and to prepare the future data engineers and data science with hand-on. This report corresponds to the second assignment for Cloud computing lab session where the aim is to understand the concepts Map reduce and Machine learning model in cloud computing world. In the Mapreduce section we first give the introduction where we describe its usefulness in the data science ecosystem and later we demonstrate how map-reduce was implemented on AWS instance. In the section for Machine learning implemented, we implement three different Machine learning models in order to give a prediction of the temperature which was later deployed in AWS.

1 Introduction

This report presents two separate results which include the MapReduce implementation using spark and Machine Learning model implementation. MapReduce is a programming model for processing big amounts of data. In the first part of the assignment we demonstrate one of the typical applications of MapReduce - calculating the number of words in a large text. We run a MapReduce program using Pyspark on a AWS instance t2.2xlarge.
In the second part of the assignment, we create a cluster of instances and run a machine learning application on the them. Namely, we use three machine learning models to predict temperature in Chinese regions.

The rest of the section of this report is organized as bellow: first we described the benchmarking steps and presents the results of how we implemetated the wordCount on AWS instance in section 2. We then describe steps follows to implement and test the Machine learning model with the provided data-sets in section 3.

2 Text processing with Pyspark

2.1 Setting up the environment for Pyspark

One of the first projects implementing MapReduce model was Hadoop. The main disadvantage of Hadoop was the amount of disk writing operation, which eventually became a bottleneck for performance.
In Spark, another framework implementing MapReduce, the calculations are done in memory, which significantly increases the speed. The loaded data is transformed into RDDs (resilient distributed datasets) and all data operations are performed on RDDs.
To implement the first task, we are using the Pyspark package, which provides API for using Spark with Python.

2.1.1 Installing Pyspark

We installed the Pyspark on t2.2xlarge instance. Here are the main steps to follow:

1. Make sure Python3, Java and Scala are installed on the instance.

2. Download the latest version of Apache Spark from the official website:
   https://spark.apache.org/downloads.html. We used spark-3.0.0-preview2-bin-hadoop2.7.

3. Extract the downloaded file:

   ```
   tar -xvf home/ubuntu/spark-3.0.0-preview2-bin-hadoop2.7
   ```

4. Set up the Spark path and Py4j path by adding the following lines to the .bashrc file:

   ```
   export SPARK_HOME = /home/ubuntu/hadoop/spark-3.0.0-preview2-bin-hadoop2.7
   export PATH = $PATH:/home/ubuntu/spark-3.0.0-preview2-bin-hadoop2.7/bin
   export PYTHONPATH = $SPARK_HOME/python:$SPARK_HOME/python/lib/
   py4j-0.10.4-src.zip:$PYTHONPATH
   export PATH = SPARK_HOME/python:PATH
   export PYSPARK\_PYTHON=/usr/bin/python3
   export PYSPARK\_DRIVER\_PYTHON=python3
   ```

5. In order to apply the added exports, launch the command:

   ```
   source .bashrc
   ```

6. From thespark-3.0.0-preview2-bin-hadoop2.7 directory, Pyspark can be launched with the command:

   ```
   ./bin/pyspark
   ```

7. If launched, we have successfully installed Pyspark.

2.1.2 Configuring S3 storage

To configure S3 storage and send data to Pyspark from it, we followed the following steps:

1. Create an S3 bucket and upload the file we want to process with Pyspark there.
2. Configure the user, who has an access to the file and get his AWS Access Key ID and AWS Secret Access Key.
3. Install aws-cli with the command:
   sudo apt install awscli.
4. Run the aws configure command to set the s3 access parameters (AWS Access Key ID, AWS Secret Access Key, location).
5. Install python boto3 library to interact between S3 and Pyspark.

After these steps we are set to write the word counting program.

2.2 Implementing word counting with Pyspark

In this section we explain the main points of the python script for counting words with Pyspark providing some excerpts from our code.

First of all, we create a SparkContext object:

```
sc = SparkContext("local", "wordcount").
```

SparkContext acts as a link between our driver (main program) and worker nodes. We specify that the local resources of the machine will be used.

Secondly we need to create an RDD, which contains the text to count the words from. We create it from the file that is stored in the S3 storage:

```
rdd = sc.textFile('result.txt').
```

At this step, each line of the text represents a single element of the RDD. Next we apply so called 'transformations' and 'actions' to our RDD to calculate the words. The 'transformations' create a new dataset from an existing one, while 'actions' return a specific values to the driver program after running a computation on the dataset. The 'transformations' are lazy, that means they aren't executed before an 'action' operation is used.

With the flatMap operation we perform a 'clean_str' function on each element of the RDD:

```
rdd = rdd.flatMap(lambda x: clean_str(x)).
```

This function takes a text line at the input, makes all the words lowercase, replaces all special characters in the beginning and the end of the line with empty string. Finally, it splits the line in places, where special characters (commas, dots, spaces, etc) occur. We implement the 'filtering' using python regular expressions:

```
def clean_str(x):
    x = x.lower()  # convert all words to lowercase
    clean_str = re.sub(r'^\W+|\W+$', '', x)  # remove special characters in the
    return re.split(r'\W+', clean_str)  # split the line
```

Then we remove the stopwords and do the word stemming. The stopwords we use are downloaded from nltk library and we also use PorterStemmer from the same library to stem the words :

```
rdd = rdd.filter(lambda x: x not in stopWords)  # remove stopwords
rdd = rdd.map(lambda x: stemWord(x))  # stemm the words
```

Now it's time for the key operations of our driver program: map and reduce. At this stage each element of our RDD represents a single stemmed word. With map operation we map each word with the value of '1'. With reduceByKey operation we group all elements with the same 'key' (the same words) and cumulatively add their values, which corresponds to the total number of the words found in the text:

```
rdd = rdd.map(lambda x: (x, 1)) #  perform map
rdd = rdd.reduceByKey(lambda x, y: x + y)  # perform reduce
```

Finally, to find the most frequent words, we exchange the places of keys and values in our 'reduceByKey' output. Now the 'keys' contain the number of times the word occurred in the text. We then use 'sortByKey' and 'take' operations to display the 20 most frequent words in the text:

```
rdd = rdd.map(lambda x: (x[1], x[0]))  # change places of keys and values
rdd = rdd.sortByKey(False)  # sort word frequencies in descending order
```

The top twenty most frequent words in the text are shown in the table 1.

Table 1: Top 20 most frequent words

Word	Frequency	Word	Frequency
one	146	see	98
would	133	gutenberg	93
said	131	project	90
like	123	time	85
look	123	seem	79
kurtz	122	thing	72
man	114	made	67
could	111	back	66
work	109	came	63
know	99	river	62

3 Building machine learning models

This section describes the process of building three machine learning models in order to predict temperature based on relevant meteorological data provided.

To build the models, we followed a set the process described in the previous work [2].

The main steps of any machine learning project include:

- Data acquisition

- Data pre-processing
- Models selection
- Model training and evaluation We performed all the steps in python using Jupyter notebook and python libraries such as sklearn, pandas, matplotlib.

3.1 Data acquisition

In this step we downloaded the data from the source provided in the assignment. It consisted from 11 datasets, corresponding to meteorological data from different Chinese regions. The meteorological parameters were collected from 1 March 2013 until 29 February 2017 for each day hourly. We created a single csv file containing all the datasets using a simple python script and saved it to s3 bucket.

3.2 Data pre-processing

Data pre-processing includes such steps as data cleaning and feature engineering. We describe this steps in further detail in the sections below.

3.2.1 Data cleaning

Firstly, we visualized the data as shown to better understand it and detect possible outliers. Figure.1 and figure.2 show the distribution of dataset values. As we can see, the dataset doesn't contain any major outlying values.

Pandas dataframe created from the dataset contains 385704 rows and 18 columns. Its first 5 elements are shown in the figure.3.

Table 2: Encoding the "wd" feature to numerical value

Original value	ESE	SSE	E	ENE	NE	NNE	SW	SSW	S	SE	NNW	NW	WNW	N	W	WSW
Encoded value	2	10	0	1	4	5	12	11	8	9	6	7	14	3	13	15

Table 3: Encoding the "station" feature to numerical value

Original value	Gucheng	Aotizhongxin	Changping	Dingling	Dongsi	Guanyuan
Encoded value	5	0	1	2	3	4
Original value	Huairou	Nongzhanguan	Shunyi	Tiantan	Wanliu	
Encoded value	6	7	8	9	10	

We could notice straightaway some missing values in the dataset. The number of missing values for each feature is shown in the figure.4. The total number of 'NaN' values is 68881. Considering the high number of entries in our dataset (385707) we decided to drop the rows with missing values.

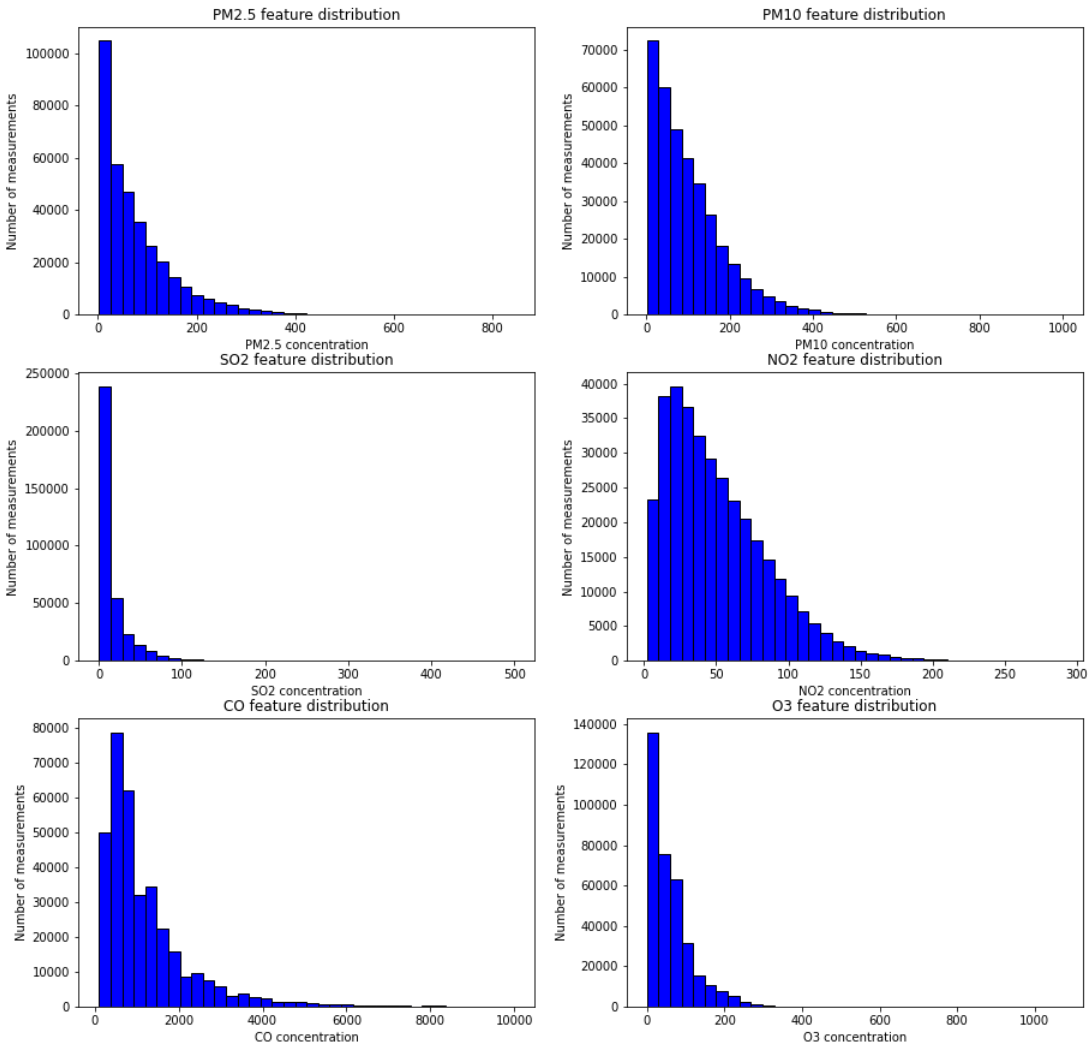

Figure 1: Data visualization

Secondly, the dataset contained categorical features with columns "wd" and "station", which we needed to convert to numerical values. One approach is to simply map each string value to numerical. We performed this using sklearn library LabelEncoder, which appointed to each string value a numerical value. The mapping is shown in table 2 and table 3.

Finally, we normilized our values using Z-score [3]. The Z score shows the number of standard deviations that the value is away from the mean. In other words:

$$scaledvalue = (value - mean)/stddev$$

. In our case, we also added a positive offset of 3 to Z-score in order to avoid having negative numbers in the training dataset, which may complicate the resulting model. So, we applied the following normalization formula for each column in our dataset:

$$scaledvalue = (value - mean)/stddev + 3$$

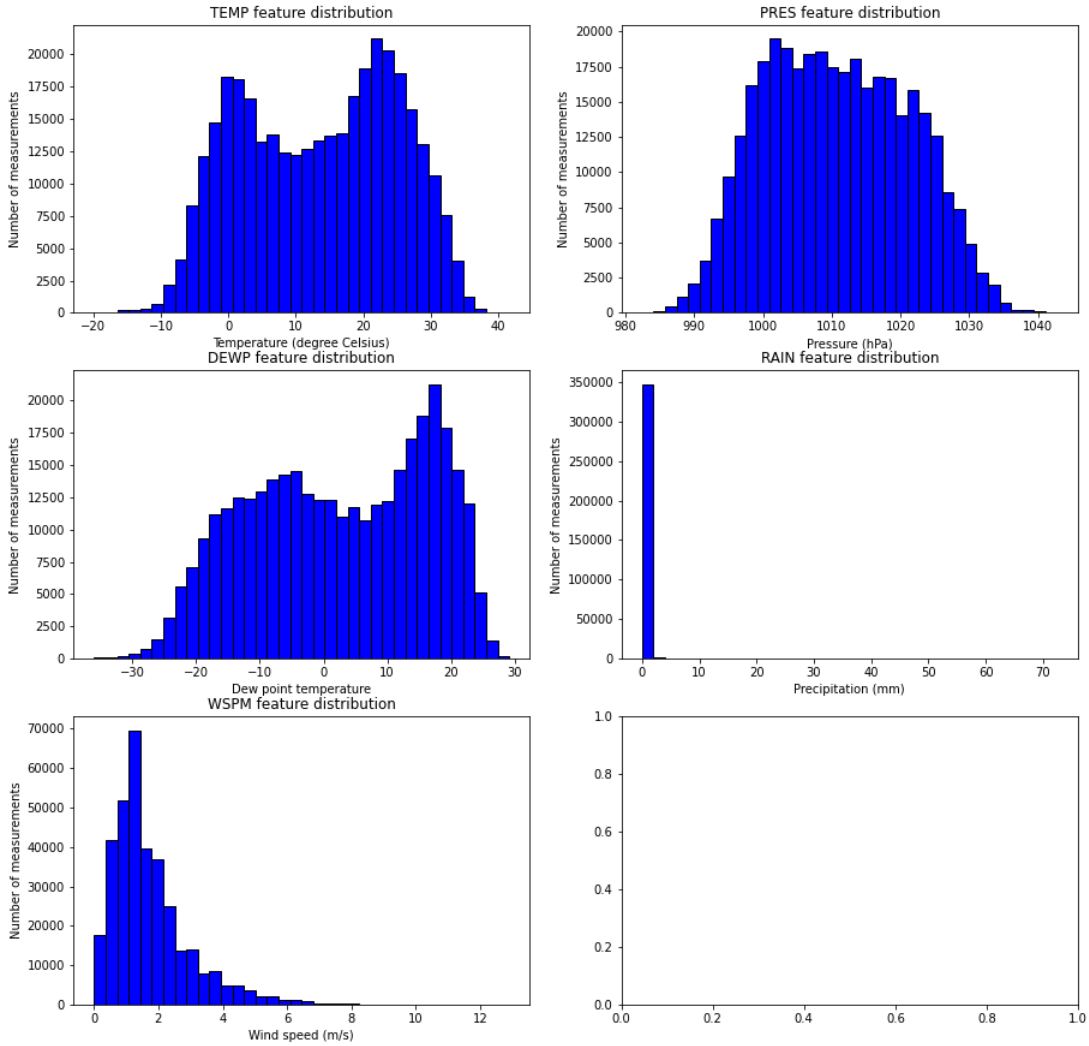

Figure 2: Data visualization (continuation)

, where 'mean' is the mean value of the column and 'stddev' - standard deviation of column values. We have also saved the normalization parameters such as mean and standard deviation of values of each column to reuse for test data normalization.

The first five elements of normalized dataframe are shown in fig.5 .

3.2.2 Feature engineering

In order to reduce the complexity and improve the performance of the model it is important to select the features related to the predicted value. We have 16 features in our dataset. In order to select the most important ones for training the model, we calculated the correlation coefficient of the features with the label (values of temperature) and visualize the results with a so called 'heat map', shown in fig.6. In the figure we can see correlation values between the labels and features, coloured according to how high the correlation is. Greener or redder colours indicate higher correlation. In our approach we chose for training the features with correlation coefficient higher,

```
# Showing First five elements.
data.head()
```

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	TEMP	PRES	DEWP	RAIN	wd	WSPM	station
0	1	2013	3	1	0	6.0	18.0	5.0	NaN	800.0	88.0	0.1	1021.1	-18.6	0.0	NW	4.4	Gucheng
1	2	2013	3	1	1	6.0	15.0	5.0	NaN	800.0	88.0	-0.3	1021.5	-19.0	0.0	NW	4.0	Gucheng
2	3	2013	3	1	2	5.0	18.0	NaN	NaN	700.0	52.0	-0.7	1021.5	-19.8	0.0	WNW	4.6	Gucheng
3	4	2013	3	1	3	6.0	20.0	6.0	NaN	NaN	NaN	-1.0	1022.7	-21.2	0.0	W	2.8	Gucheng
4	5	2013	3	1	4	5.0	17.0	5.0	NaN	600.0	73.0	-1.3	1023.0	-21.4	0.0	WNW	3.6	Gucheng

Figure 3: Showing first five elements of the dataset

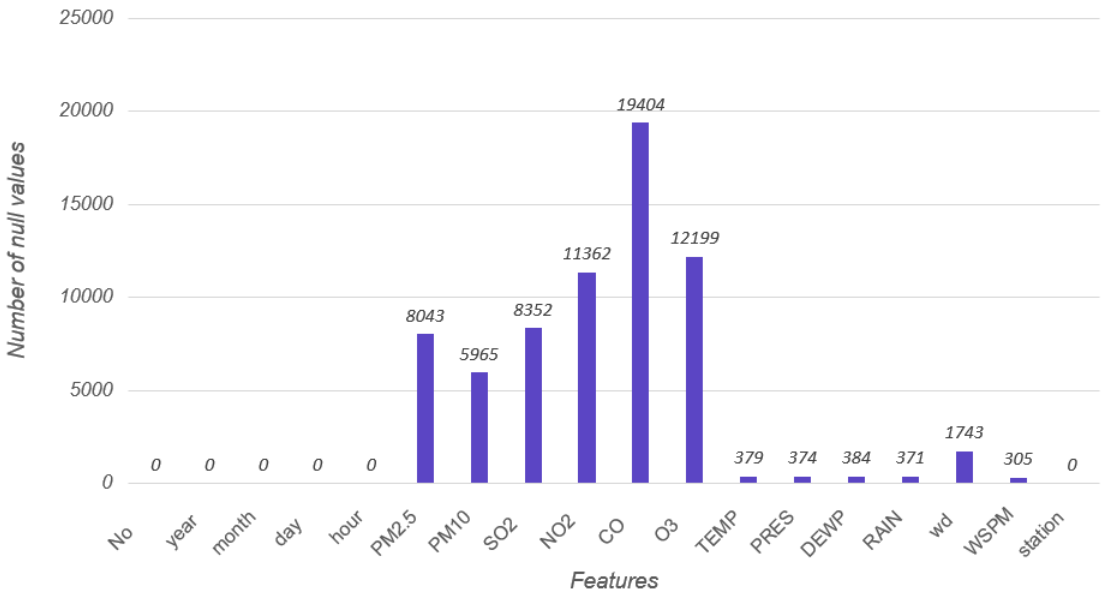

Figure 4: Number of null values in the dataset

	No	year	month	day	hour	PM2.5	PM10	SO2	NO2	CO	O3	TEMP	PRES	DEWP	RAIN	wd	WSPM	station
20	21	1.520044	1.993252	1.328930	4.214379	2.172252	2.131614	2.834133	1.709557	2.897836	3.061524	1.6	4.566715	1.493421	2.921099	1.956500	3.131490	3.003261
21	22	1.520044	1.993252	1.328930	4.358594	2.197342	2.109597	2.929118	1.938330	2.984780	2.902728	1.0	4.662500	1.565855	2.921099	3.724852	2.167330	3.003261
22	23	1.520044	1.993252	1.328930	4.502809	2.209887	2.164639	3.024104	2.109909	2.984780	2.779220	1.3	4.691236	1.551368	2.921099	1.514412	2.408370	3.003261
23	24	1.520044	1.993252	1.328930	4.647024	2.209887	2.164639	2.929118	2.138506	2.897836	2.761576	0.2	4.710393	1.551368	2.921099	1.956500	2.488717	3.003261
24	25	1.520044	1.993252	1.442516	1.330081	2.184797	2.054556	3.404045	2.796227	3.157798	2.426340	-0.3	4.739129	1.522395	2.921099	1.735456	2.729757	3.003261

Figure 5: Showing first five elements of the normalized dataset

than 9 %. More specifically, we chose such features as "year", "month", "hour", "PM2.5", "PM10", "SO2", "NO2", "CO", "O3", "PRES", "DEWP" and "wd".

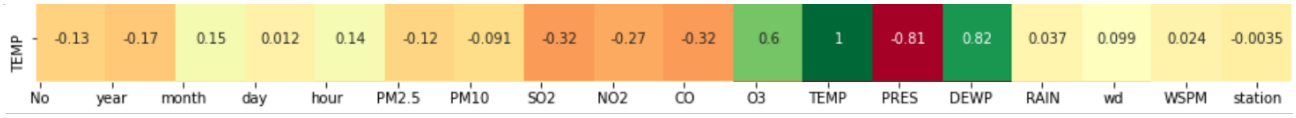

Figure 6: Correlation coefficients between label and features

3.3 Models selection

The model selection phase is closely related with the dataset and the objective of the task. This phase was done in parallel with data discovery and analysis, in order to find the most appropriate models that would solve the problem.

3.3.1 Problem definition and model selection method

Firstly we needed to identify the type of problem we had, its objective, the input and the expected output. As shown in Figure 7 , there are many different machine learning algorithms that can be grouped under different techniques depending on how the algorithm works.

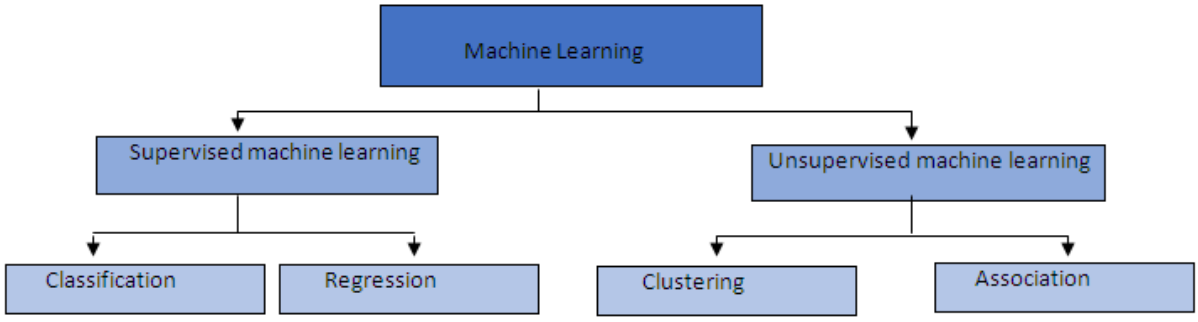

Figure 7: Main machine learning techniques

Supervised learning methods involves the training of the system where the labled training set is provided to the system for performing a task.

Unsupervised Learning does not involve the target output which means the system isn't trained.The system has to learn by its own through determining and adapting according to the structural characteristics of the input labels. It is where we provide only unlabeled input data. As described in the assignment, the goal was to build machine learning models that would predict temperature based on some given parameters. As we already had a set of meteorological data and corresponding output variables (temperature), the best solution for this kind of problem is **supervised machine learning**. We then had to choose between classification models and regression models. The main difference between them is that the output variable in regression is numerical (or continuous) while that for classification it is categorical (or discrete). The provided dataset contains numerical variables, but the assignment specified that the predicted results should be categorical variables (from verycold to veryhot). Both algorithms (Classification and Regression) fit our problem, as we could just categorize the output variable from the original dataset before training the model, or train the model with continuous variables, then convert them to categorical variables.

To make our decision, we trained different classifier models and the equivalent regression models that are available from the main sklearn modules. Those models are presented in Table 4. Each of these models were then tested and we compared their accuracy. Figure 8 shows the results we got. We find that Regressors relatively outperfom Classifiers, and the models with the best accuracy are sklearn.ensemble.RandomForestRegressor (accuracy = 0,96) and

Table 4: Different models tested

Module	Classifier	Regressor
sklearn.multioutput	MultiOutputClassifier	MultiOutputRegressor
sklearn.neighbors	KNeighborsClassifier	KNeighborsRegressor
sklearn.ensemble	RandomForestClassifier	RandomForestRegressor
sklearn.linear_model	SGDClassifier	SGDRegressor
sklear.tree	DecisionTreeClassifier	DecisionTreeRegressor

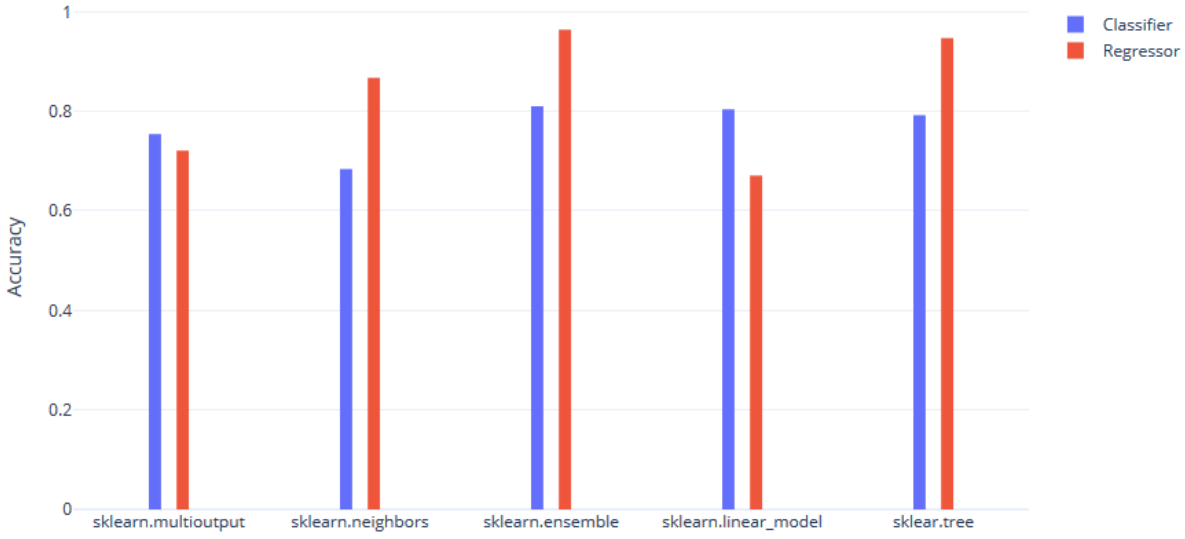

Figure 8: Accuracy of different machine learning models

sklearn.tree.DecisionTreeRegressor (accuracy = 0,94). These models were thereby chosen. For the svm model, we choosed the regressor model to go with the ones we have already chosen. Better results for regression models can be explained by the fact that the correlation coefficients for dataset with categorized temperature values (fig.9 is lower, than for categorized label values (fig.6).

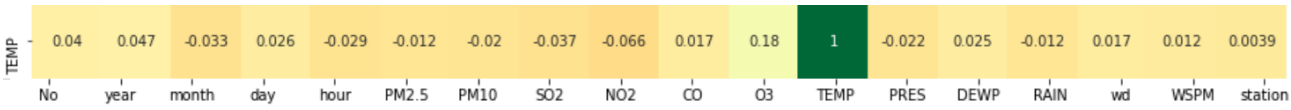

Figure 9: Correlation coefficients between categorized labels and features

Finally, he chosen models are:

- First model : Support Vector Regression from the Support Vector Machines module sklearn.svm.SVR.
- Second model : RandomForestRegressor from the ensemble module sklearn.ensemble.DecisionTreeRegressor.

- Third model : DecisionTreeRegressor from the tree module sklearn.tree.DecisionTreeRegressor.

3.4 Model training and evaluation

The training and evaluation phase let us train the model and evaluate how well it is able to map inputs to outputs and make accurate predictions. It involves three main steps: (1) Splitting the pre-processed data intro training, testing and/or validation set, (2) Defining and training the model with parameter tuning, (3) Testing and validating the model.

3.4.1 Test train split

In order to be able to estimate the model performance on unknown dataset, in machine learning the dataset is first split into training dataset and testing dataset. There is no rule as to the exact split size, but it is sensible to reserve a larger sample for training, for instance 80% for training and 20% for testing data. Also, for even better overfitting prevention, the train dataset is split in actual training dataset and validation dataset. Validation dataset will be used to evaluate the performance of the model in order to 'tune' it afterwards. Again, there are no rules for the splitting proportion, but it's recommended to leave more data for training dataset [4]. In the end, the test dataset is used for final model performance evaluation. This dataset isn't biased, as it hasn't seen the model before. So, we split our dataset in the following proportion: 60% for training, 20% for validation and 20% for test data.

3.4.2 Hyperparameter tuning

A hyperparameter is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training. Different model training algorithms require different hyperparameters, that are specific to the mathematical function used by the model. The same kind of machine learning model can require different hyperparameters (constraints, weights or learning rates) to generalize different data patterns. These measures have to be tuned so that the model can optimally solve the machine learning problem.

Hyperparameter optimization or tuning is the process of choosing a set of optimal hyperparameters for a learning algorithm. It finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given data. In our case, we used GridSearch to perform the tuning. It is an exhaustive searching through a manually specified subset of the possible hyperparameters of the model. The goal of our tuning function is to find the best set of hyperparameters that maximize the score (accuracy) of the model. This was done by using the sklearn.model_selection.GridSearchCV function.

3.4.3 First model sklearn.svm.SVR

Support vector regression (SVR) is a statistical method that examines the linear relationship between two continuous variables. In regression problems, one generally tries to find a n-dimensional space that best fits the data provided. In the case of regression using a support vector machine, we do something similar but with a slight change. Here we define a small error value e (error = prediction - actual). The value of e determines the number of support vectors, and a smaller e

value indicates a lower tolerance for error. In sklearn, this error is called 'epsilon', and it is one of the hyperparameters that we can tune to improve our model. Other main hyperparameters are:

- C : The C parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words, C behaves as a regularization parameter.

- kernel : kernel is a set of mathematical functions. Kernel function takes data as input and transforms it into the form required by the output. In sklearn, the only kernels available are 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.

- gamma : the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

For this model, the best parameters we got through hyperparameter tuning are C=1.0, epsilon=0.1, gamma='scale' and kernel='rbf'. For the other parameters, we used the default provided by sklearn. Sklearn provides a simple function that we used for training the model. The code is as follows: model.fit(training_dataset).

3.4.4 Second model sklearn.ensemble.RandomForestRegressor

Random forest is a type of supervised machine learning algorithm where we join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithms of the same type i.e. multiple decision trees, resulting in a forest of trees, hence the name: "random forest". The Random Forest algorithm executes a number of steps to get the final output.

Firstly, it picks N random records from the dataset and build a decision tree based on these N records. In the second step, it chooses the number of trees we want in the algorithm and repeats the first step. Each tree in the forest predicts a value for Y (output) in step 3. The final value can be calculated by taking the average of all the values predicted by all the trees in forest. This model uses many hyperparameters, that also affect the training time and the size of the final model. The main hyperparameters we tuned are:

- n_estimators : it is the number of trees in the forest. Ideally, one can expect a better performance from the model when there are more trees. However, increasing the number of trees too much can cause the model to be extremely slow to train and to predict, and will need lot of computation resources and memory, while not achieving better results. The GridSearchCV function we used returned 100 as the best value for n_estimators

- max_depth : It is the maximum depth of the trees. It sets the maximum number of node each tree of the algorithm can expand when when learning from the data. The value returned by the tuning function is 'None'. When max_depth s set to 'None', nodes are expanded until all leaves are pure or until all leaves contain less than the number of random records N picked in the first step of the algorithm.

For all the other hyperparameters, we used the default values provided by sklearn. The list of all the possible parameters can be found at [1].

3.4.5 Third model: sklearn.tree.DecisionTreeRegressor

A 'Decision tree' is a flowchart like a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has the same value of the target variable, or when splitting no longer adds value to the predictions.The main hyperparameters we tuned are:

- criterion: this function is used to measure the quality of a split in the decision tree regression. By default, it is 'mse' (the mean squared error), and it also supports 'mae' (the mean absolute error).
- max_depth: this is used to add maximum depth to the decision tree after the tree is expanded.
- min_samples_leaf: this function is used to add the minimum number of samples required to be present at a leaf node.

The values returned by the tuning function for these parameters are criterion='mse', max_depth=None (when set to None, nodes are expanded until all leaves are pure or until all leaves contain less than the number of samples) and min_samples_leaf=1

3.4.6 Results and evaluation

All the models were created using the hyperparameters provided by the tuning function and the other default value provided by sklearn as shown in Table 5.

The models were then trained using the model.fit(train_data) function and saved using the python pickle module. Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object. The models were saved as follows:

- model1.sav : sklearn.svm.SVR
- model2.sav : sklearn.ensemble.RandomForestRegressor
- model3.sav : sklearn.tree.DecisionTreeRegressor

The models were evaluated using k-fold cross validation. **Cross validation** is an approach used to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts. For modest sized datasets in the thousands or tens of thousands of records, k values of 3, 5 and 10 are common. We choosed

Table 5: Creating the models with the hyperparameters provided by the tuning function

Model name	Model parameters
sklearn.ensemble.RandomForestRegresso	n_estimators=100, crite- rion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, ran- dom_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None
sklearn.tree.DecisionTreeRegressor	criterion='mse', split- ter='best', max_depth=None, min_samples_split=2, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort='deprecated', ccp_alpha=0.0
sklear.svm.SVR	kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1

k = 3 and splitted our dataset into three parts: Training set, validation set and test set with training set = 60 % of the original pre-processed dataset, validation set = 20% and testing set = 20%. The algorithm was trained different parts of the training dataset, and the model was then tested with validation dataset. After applying rounds of evaluation with different parts of the train dataset and final testing with test dataset we ended up with two different performance scores that we summarized in table 6 using a mean and standard deviation. The result is a more reliable estimate of the performance of the algorithm on new data given our test data. It is more accurate because the algorithm is trained and evaluated multiple times on different data.

Table 6: Accuracy score of the models

Model	Average score (%)	standard deviation
sklearn.ensemble.RandomForestRegressor	96,14	0,68
sklearn.tree.DecisionTreeRegressor	94,52	0.43
sklear.svm.SVR	95,33	1,02

The table shows that RandomForestRegressor achieves the best result, but the other models achieve good results too that are quite acceptable, given the original problem.

4 Running machine learning models on AWS EMR Cluster

In order to train our models, we set up the AWS EMR cluster, which consisted of three instances: one master instance and two slaves instances. The master is responsible for distributing the load between the slaves. We used m4.large instances to run in the cluster. In order to submit a job to the cluster it's necessary to connect to the master node and launch the python script on it. For connection we are using SSH. One important point is to configure the firewall of the master node with the inbound rule to allow SSH. So, the inbound rule that needs to be added is shown in table .7.

Table 7: Firewall inbound rule to enable SSH

Type	Protocol	Port	Source	Rule
SSH	TCP	22	24.200.159.131/32	Allow

In the script we implement dataset upload and download from S3 using boto3 python library.

After all our models were trained, tested and saved, we then uploaded them to our amazon s3 bucket. In order to make them accessible by people not having the credentials, we made them public. We then made a program named predict_methods.py that can be run on EMR cluster. The program reads a csv input dataset from a s3 bucket, applies the three models to the datasets and saves the result back to the provided s3 bucket. Follow the steps below to configure and run the program.

1. First make sure you have a s3 bucket and a EMR cluster configured with pyspark, up and running. You can create new ones or use existing ones.

2. Open the file predict_methods.py and fill the necessary parameters:

    - Set your s3 bucket credentials access key id (parameter name is aws_key_id and secret access key (parameter name is aws_secret_key. This will let the code get your dataset from your s3 and upload the results back there. If you don't have these credentials, please follow the first part of this tutorial to get them https://realpython.com/python-boto3-aws-s3
    - Set the s3_bucket parameter with your amazon s3 bucket name
    - Set the input_name parameter with the name of the csv dataset file hosted on your s3 bucket.
    - Set the output_name with the name of your desired result file name. Note that the output file name will be used to save the results of the prediction for all the models and should be a csv file.

- The models url are already provided with parameters model1, model2 and model3. model1 is the url of sklearn.svm.SVR, model2 is for sklearn.ensemble.RandomForestRegressor and model3 is for the DecisionTreeRegressor. The keys of the models on our s3_bucket have the same name as the model file name (model1.sav, model2.sav and model3.sav)

3. Save and upload the file to your emr cluster's master node, then run python3 predict_methods.py to execute the code. If the code doesn't start, make sure the file has the executable right by running chmod +x predict_methods.py, then execute the code.

4. The program will automatically install the required libraries, download your dataset, predict new values and upload the results to your s3 bucket. The results are categorized from verycold to veryhot according using scale given by the assignment and saved to a csv file. That file (which filename will be the value of output_name parameter), have four columns:

   - Actual: The (categorised) true temperature from the dataset
   - model1_pred: The (categorised) predicted temperature using model 1
   - model2_pred: The (categorised) predicted temperature using model 2
   - model3_pred: The (categorised) predicted temperature using model 3

5. You should pay attention to the shell output when the code will be running, as it will provide valuable information on each step of the code and the accuracy score of each of the models.

5 Running the scripts and retrieving the models

To run the script for counting words with Pyspark, you should enter the S3 bucket parameters in the script (the fields will be indicated) and then run the script with the command "pyhton3 word_count.py".

To run the script for using the models, you should also enter S3 parameters as indicated in the script and run the command "python3 predict_methods.py".

The trained models can be retrieved using the link:

https://drive.google.com/file/d/1cUA1sKmyGRsB8yvojMCHZYqUkEp7d6SL

6 Conclusion

We report the results of our analysis for text processing by implementing a wordCount algorithms using Spark and temperature prediction using classification model. The steps followed was guideline by the set of the instructions provided in assignment and some additional recent works with clear references where necessary. For example in addition to stopwords removal to reduce noise in our dataset, we chose to also stemmed word to avoid the duplicate word counting. The result for the word count implementation shows the word "one" with the highest occurrence while "river" has the lowest, with the frequencies 146 and 62 respectively. To predict the temperature range we implemented three separate regression models each giving the high accuracy ranging from 94.52 to 96.14.

References

[1] scikit learn, https://scikit-learn.org, accessed on 2020-03-20.

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300. IEEE, 2019.

[3] S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[4] Z Reitermanova. Data splitting. In *WDS*, volume 10, pages 31–36, 2010.