Paralleling Linear Regression in Hadoop MapReduce

Jinghua Feng fengj3@rpi.edu

1. Hadoop installation on Linux

1.1 Installation

Hadoop 2.7.5 binary version was downloaded from Apache Hadoop[1]. Followed the instructions by K Hong[2], installation of Hadoop is processed step by step on Ubuntu 14.04. But there are two main changes needed to be pointed out.

First, as newest version of java (1.8.0_161, released on Jan 16, 2018) had been installed before Hadoop, the step of installing java is not necessary. In addition, when setting up configuration files, the environment variable JAVA_HOME should be set as "/usr/lib/jvm/java-8-oracle" instead of "/usr/lib/jvm/java-7-openjdk-amd64".

Second, for the setup of "~/.bashrc", as below figure shows, the path of last environment variable "HADOOP_OPTS" should be "-Djava.library.path=$HADOOP_INSTALL/lib/native" instead of "-Djava.library.path= $HADOOP_INSTALL/lib". This is a common trouble that a lot of people encountered when installing Hadoop.

```
hduser@liuelab:/usr/local/hadoop/lib/native$ ll
total 4380
drwxr-xr-x 2 hduser hadoop    4096 Feb 11 19:23 ./
drwxr-xr-x 3 hduser hadoop    4096 Feb 11 19:23 ../
-rw-r--r-- 1 hduser hadoop 1123566 Dec 15 20:12 libhadoop.a
-rw-r--r-- 1 hduser hadoop 1486932 Dec 15 20:12 libhadooppipes.a
lrwxrwxrwx 1 hduser hadoop      18 Dec 15 20:12 libhadoop.so -> libhadoop.so.1.0.0*
-rwxr-xr-x 1 hduser hadoop  673620 Dec 15 20:12 libhadoop.so.1.0.0*
-rw-r--r-- 1 hduser hadoop  581856 Dec 15 20:12 libhadooputils.a
-rw-r--r-- 1 hduser hadoop  364916 Dec 15 20:12 libhdfs.a
lrwxrwxrwx 1 hduser hadoop      16 Dec 15 20:12 libhdfs.so -> libhdfs.so.0.0.0*
-rwxr-xr-x 1 hduser hadoop  229193 Dec 15 20:12 libhdfs.so.0.0.0*
```

Figure 1 Path of libhadoop and libhdfs

1.2 Test HDFS and MapReduce examples

Hadoop infrastructure contains two main components: storage and processing[3], which are realized by HDFS and MapReduce respectively. To guarantee Hadoop run successfully and understand the working procedures better, HDFS common commands (ls, cat, copy, …) for managing files and MapReduce example are tested, basically following instructions of K Hong[4].

1.2.1 HDFS

Run the following commands one by one and obtain the same results

Create Home directory in HDFS

hduser@liuelab:~$ hdfs dfs -mkdir -p /user/hduser

Create a test file “hdfs-test.txt” in local disk

hduser@liuelab:~$ echo "hdfs test" > hdfs-test.txt

Copy file hdfs-test.txt from local disk to HDFS user’s directory in HDFS

hduser@liuelab:~$ hdfs dfs -copyFromLocal ~/hdfs-test.txt hdfs-test.txt

Get a directory listing of the user's home directory in HDFS

hduser@liuelab:~$ hdfs dfs -ls

Display the contents of the HDFS file /user/hduser/hdfs-test.txt:

hduser@liuelab:~$ hdfs dfs -cat /user/hduser/hdfs-test.txt

Copy that file to the local disk from HDFS, named as hdfs-test.txt :

hduser@liuelab:~$ hdfs dfs -copyToLocal /user/hduser/hdfs-test.txt hdfs-test2.txt

1.2.2 MapReduce

We run the example for pi calculation, below picture shows the results in the Ubuntu 14.04 platform, which is consistent with the results shown in the instructions[4].

```
hduser@liuelab:/usr/local/hadoop$ hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.5.jar pi 2 5
Number of Maps  = 2
Samples per Map = 5
Wrote input for Map #0
Wrote input for Map #1
Starting Job
Job Finished in 1.474 seconds
Estimated value of Pi is 3.60000000000000000000
hduser@liuelab:/usr/local/hadoop$ hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.5.jar pi 16 1000
Number of Maps  = 16
Samples per Map = 1000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Wrote input for Map #10
Wrote input for Map #11
Wrote input for Map #12
Wrote input for Map #13
Wrote input for Map #14
Wrote input for Map #15
Starting Job
Job Finished in 2.507 seconds
Estimated value of Pi is 3.14250000000000000000
```

1.3 Test Hadoop streaming with Python

MapReduce framework contains three major phases-map, shuffle and sort, and reduce. The Mapper function in the first phase sequentially processes a series of key-value pairs and produces zero or more key-values pairs as output. The shuffle and sort phase sorts the intermediate keys and values for each partition, and moves mapper output to reducer (shuffling). In the third phase, using the output of mapper as input, the reducer function aggregates the values for each unique key and produces zero or more output key-value pairs[5].

Hadoop is mostly written in Java, but we prefer to program in Python. Therefore, we have to learn how to use Python in Hadoop. Fortunately, Hadoop streaming utility enables Python, shell scripts, or any other language to be used as a mapper, reducer, or both[5].

Following the steps of Michael's tutorial[6], we tested the word count example on this website. The only minor change is the path of the Hadoop streaming jar file. The default path on ubuntu 14.04 is "share/hadoop/tools/lib/hadoop-streaming-2.7.5.jar", but not "contrib/streaming/ hadoop-*streaming*.jar". The first counted ten words are shown in below figure, which are exactly the same with the tutorial.

```
hduser@liuelab:/usr/local/hadoop$ hdfs dfs -cat guten*t/par* | head -10
"(Lo)cra"       1
"1490   1
"1498," 1
"35"    1
"40,"   1
"A      2
"AS-IS".        1
"A_     1
"Absoluti       1
"Aesopi"        1
```

2. Multi-Variate linear regression theory

If there are n data points and m descriptive features in dataset, we can use X_{nm} to denote data matrix, and $\vec{y}_n$ to denote response vector. Regression coefficient vector (weights) is represented by $\vec{w}_m$. Then we have the following equation,

$$X_{nm}\vec{w}_m = \vec{y}_n$$

Multiplying the transpose of data matrix X_{nm} to both sides,

$$X_{nm}^T X_{nm}\vec{w}_m = X_{nm}^T\vec{y}_n$$

Multiplying the inverse of $X_{nm}^T X_{nm}$,

$$(X_{nm}^T X_{nm})^{-1}(X_{nm}^T X_{nm})\vec{w}_m = (X_{nm}^T X_{nm})^{-1}\vec{y}_n$$

$$\vec{w}_m = (X_{nm}^T X_{nm})^{-1}\,(X_{nm}^T\vec{y}_n) \qquad (1)$$

The equation (1) is equivalent to the ordinary least-squares method.

Descale weights

It is customary to standardize or scale the data before applying linear regression model. The typical Z-score scaling method is to subtract the average of variable from variable itself, then divide the difference by standard deviation. This implies that the weights we obtain from equation (1) are the solution of scaled data. Thus, we should rescale the weights to obtain the correct solution for raw data.

For raw data, we use $x_i, w_i, \text{and } y$ to denote the i-th variable, the corresponding weight, and response. w_i' represents the scaled weight for i-th variable. We have the following two equations,

$$\sum_{i=1}^{m} w_i x_i = y \tag{2}$$

$$\sum_{i=1}^{m} w_i' \frac{x_i - \bar{x_i}}{\sigma_{x_i}} = \frac{y - \bar{y}}{\sigma_y} \tag{3}$$

Rearrange equation (3), we can obtain,

$$\sum_{i=1}^{m} \frac{\sigma_y}{\sigma_{x_i}} w_i' x_i + (\bar{y} - \sum_{i=1}^{m} \frac{\sigma_y}{\sigma_{x_i}} w_i' \bar{x_i}) = y \tag{4}$$

Based on equation (2) and (4),

$$w_i = \frac{\sigma_y}{\sigma_{x_i}} w_i' \tag{5}$$

The constant term should be incorporated as w_0

$$w_0 = \bar{y} - \sum_{i=1}^{m} \frac{\sigma_y}{\sigma_{x_i}} w_i' \bar{x_i} \tag{6}$$

3. Linear regression in MapReduce

MapReduce framework contains three major phases-map, shuffle and sort, and reduce. The Mapper function in the first phase sequentially processes a series of key-value pairs and produces zero or more key-values pairs as output. The shuffle and sort phase sorts the intermediate keys and values for each partition, and moves mapper output to reducer (shuffling). In the third phase, using the output of mapper as input, the reducer function aggregates the values for each unique key and produces zero or more output key-value pairs[5].

Therefore, the crucial part of solving the problem is to break up the calculation in equation (1) and parallel the computation. The matrices X_{nm} and $\vec{y}_n$ can be broken into submatrices as follows,

$$X_{nm} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_n \end{bmatrix} \qquad \vec{y}_n = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

where X_i is a vector consisting of m elements, y_i is a scalar. Thus,

$$X_{nm}^T X_{nm} = [X_1^T\ X_2^T\ X_3^T \cdots X_n^T] \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_n \end{bmatrix} = [X_1^T X_1 + X_2^T X_2 +\ X_3^T X_3 \cdots X_n^T X_n] \tag{7}$$

$$X_{nm}^T \vec{y}_n = [X_1^T\ X_2^T\ X_3^T \cdots X_n^T] \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = [X_1^T y_1 + X_2^T y_2 +\ X_3^T y_3 \cdots X_n^T y_n] \tag{8}$$

3.1 Training data by MapReduce

3.1.1 Mapper

To obtain weights in equation (1), equation (2) and (3) are used to compute $\mathrm{X_{nm}^T X_{nm}}$ and $X_{nm}^T \vec{y}_n$ respectively. In equation (2), each item $X_i^T X_i$ is a $m \times m$ matrix, whose element with indexes (k, l) is $x_{ik} \cdot x_{il}$. Here (k, l) can be set as the key of the mapper output, $x_{ik} \cdot x_{il}$ is the corresponding value. $x_{1k} \cdot x_{1l}, x_{2k} \cdot x_{2l}, \cdots,\ x_{ik} \cdot x_{il}, \cdots, x_{nk} \cdot x_{nl}$ share the same key (k, l). In equation (3), each item $X_i^T y_i$ is a $m \times 1$ matrix, whose element with index k is $x_{ik} \cdot y_i$. The items with key k are comprised of $x_{1k} \cdot y_1$, $x_{2k} \cdot y_2 \cdots$, $x_{ik} \cdot y_i$, $\cdots$, and $x_{nk} \cdot y_n$.

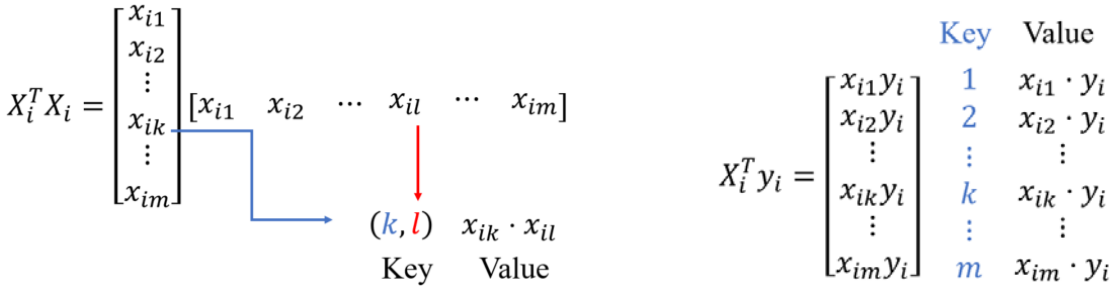

Figure 2 key-value pairs produced by mapper

Reading the input data line by line $(1,2,\cdots,\mathrm{n})$, mapper outputs for $X^T X$ and $X^T y$ are shown in left part of Figure 3 and Figure 4, respectively. Then Hadoop shuffle and sort the mapper output automatically by the keys. The sorted results are shown in right part of Figure 3 and Figure 5, which will be utilized as the input of reducer.

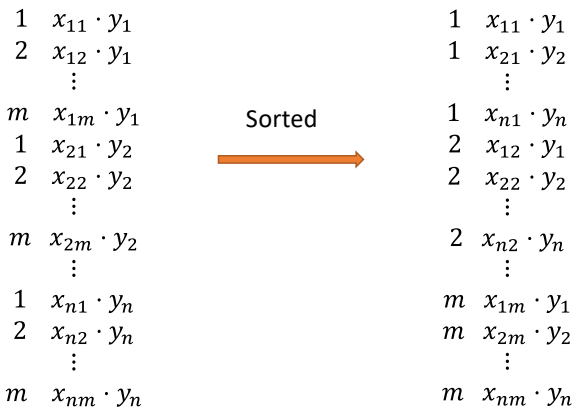

Figure 3 mapper output and sorted result for $X^T y$

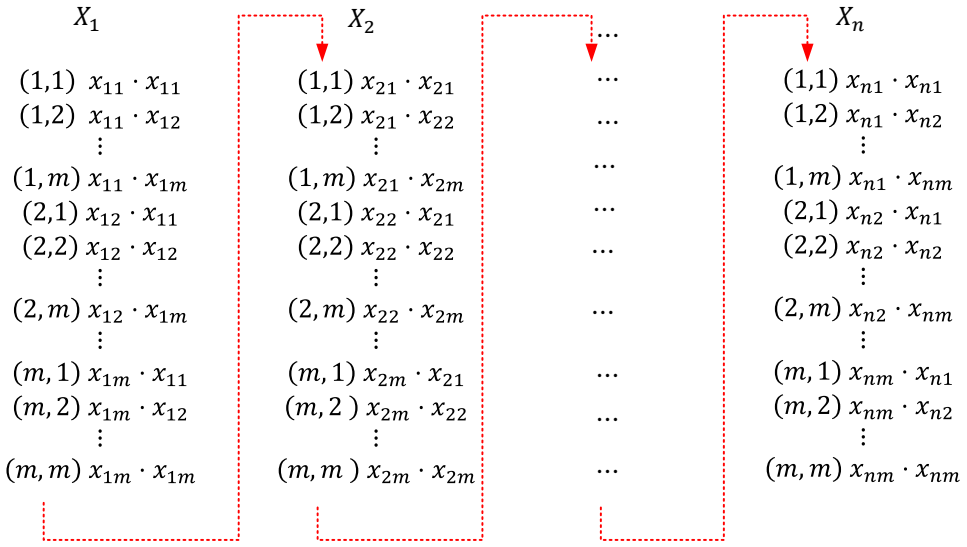

Figure 4 Mapper output for for $X^T X$

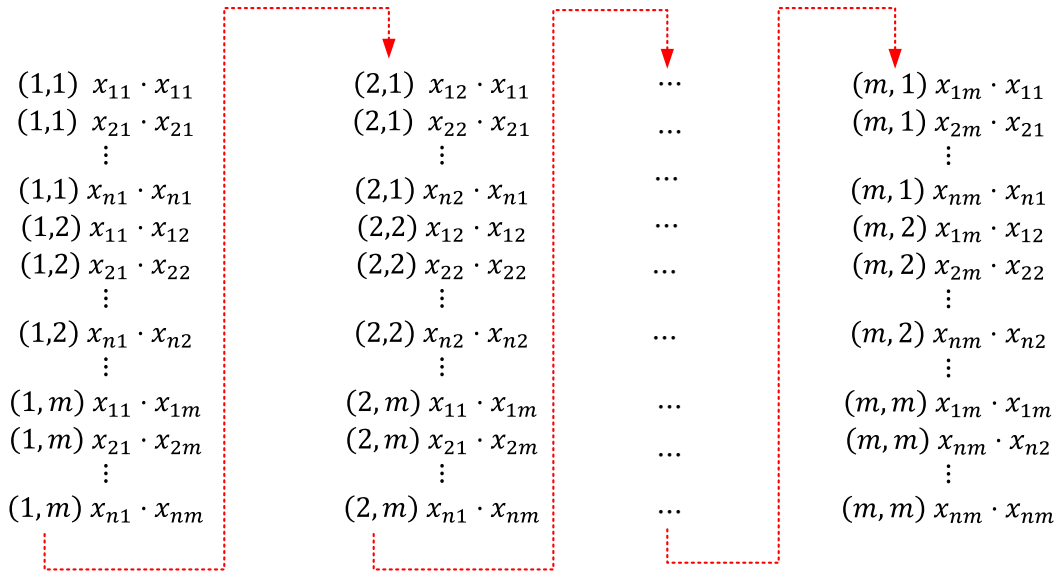

Figure 5 Shuffled and sorted data for $X^T X$

3.1.2 Reducer

Using the sorted mapper output as input, the partitioner can distribute the data uniformly to different reducers. The records with same key are grouped together and it is guaranteed that the key-value pairs in the same group are sent to the same reducer. Each reducer can sum up the values which share the same key and output the new key-value pairs with the original key and aggregated value. The new values are the entries of $X^T X$ and $X^T y$ respectively. Their corresponding keys are the index of these entries in the matrix. As we have obtained $X_{nm}^T X_{nm}$ and $X_{nm}^T \vec{y}_n$, weights of linear regression can be computed by using equation (1).

3.2 Testing data by MapReduce

3.2.1 Mapper

Using weights obtained from training data, mapper in testing phase computes predicted response. The predicted and target responses are outputted, as the input of testing reducer. Besides that, the averages of target and predicted responses are computed and written in a file called “test_ave.txt”. These two average values will be used to compute metrics (q^2 and Q^2) in reducer.

3.2.2 Reducer

In reducer, using the equations (4), (5), (8), and (9) below, metrics including mean square error (MSE), mean average error (MAE), q^2 , and Q^2 are computed. The notations in these equations are described briefly here: y^k-actual response of record k, $\hat{y}^k$- predicted response, $\bar{y}$ is the average of all actual responses, $\bar{\hat{y}}$ is the average of all predicted responses. y^k and $\hat{y}^k$ pairs are from individual output lines of mapper, while $\bar{y}$ and $\bar{\hat{y}}$ are read from file "test_ave.txt" generated by mapper.

$$MAE = \frac{1}{n}\sum_{k=1}^{k=n}|\hat{y}^k - y^k| \tag{9}$$

$$MSE = \sqrt{\frac{1}{n}\sum_{k=1}^{k=n}(\hat{y}^k - y^k)^2} \tag{10}$$

$$r^2 = \frac{\left[\sum_{k=1}^{k=n}(\hat{y}^k - \bar{\hat{y}})(y^k - \bar{y})\right]^2}{\sum_{k=1}^{k=n}(\hat{y}^k - \bar{\hat{y}})^2 \sum_{k=1}^{k=n}(y^k - \bar{y})^2} \tag{11}$$

$$R^2 = 1 - \frac{\sum_{k=1}^{k=n}(y^k - \hat{y}^k)^2}{\sum_{k=1}^{k=n}(y^k - \bar{y})^2} \tag{12}$$

$$q^2 = 1 - r^2 \tag{13}$$

$$Q^2 = 1 - R^2 \tag{14}$$

4. Dataset preparation

4.1 Dataset description

We collected weather and crime data of the same city Chicago in the whole year of 2017. Details of these two datasets and the combined one are described below.

1) Weather data

The weather data was obtained from National Oceanic and Atmospheric Administration (NOAA) [1]. The features contained in weather dataset are **Temperature, Precipitation and Snow**. It contains 365 records (one record per day). Other attributes include station collecting the data,

location (latitude, longitude), Chicago elevation, which are all fixed attributes and will be removed.

(2) Crime data

Crime data is collected from Chicago Data Portal[2]. The size of raw crime dataset is relatively much larger. It contains 4 features and 266635 tuples. Attributes are **"Community Area", "FBI Code", "Domestic" and "Arrest**" (*Please See Appendix A for the explanation of these features*).

(3) Goal of this assignment

In this assignment, binary classification for "**Arrest**" (whether the criminal is arrested) is conducted based on the combined weather and crime data features (**Temperature, Precipitation, Snow, Community Area, FBI Code, and Domestic**). DMak software developed by Professor Mark Embrechts is applied for linear regression and logistic regression.

4.2 Data preprocessing

The basic steps of data preprocessing include cleaning weather data, cleaning crime data, combining the weather and crime datasets, and adjusting the combined data for the input format of DMak software.

4.2.1 Weather data cleaning

As below Figure 6 shows, python package 'pandas' is used to read the weather data into data frame 'weather_df'. As 'TMAX' and 'TMIN' are sort of correlated with each other, a median value TEMP is added as a new attribute, replacing 'TMAX' and 'TMIN'. Then only columns of 'DATE', 'TEMP', 'PRCP', and 'SNOW' are retained in 'weather_df'. 'DATE' is kept for the merging item to combine weather and crime data later. For the consistency, the attributes are named in lower case, with the first character upper case. At last, the 'Date' is converted into standardized datetime in python by the data frame object 'to_datetime'. After running the above procedures, the resulted new 'weather_df' is shown in Figure 7.

```
# Read weather data
weather_df = pd.read_csv( "../data/raw/weather.csv" , sep = "," , parse_dates = ['DATE'] )

# Add Column 'TEMP', median value of TMAX and TMIN
weather_df[ 'TEMP' ] = ( weather_df[ 'TMAX' ] + weather_df[ 'TMIN' ]) / 2.0

# Select attributes: DATE, TEMP, PRCP, SNOW
weather_df = weather_df[ [ 'DATE' , 'TEMP' , 'PRCP' , 'SNOW' ] ]

# Change column name
weather_df.columns = [ 'Date' , 'Temp' , 'Prcp' , 'Snow' ]
weather_df[ 'Date' ] = pd.to_datetime( weather_df.Date )

# Output cleaned weather data to csv file
weather_df.to_csv( '../data/cleaned/cleaned_weather.csv' )
```

Figure 6 Python script for cleaning weather data

```
>>> import clean_data as cd
Read crime data done!
Select crime attributes done!
Sorted crime date by date done!
Removed time from datetime done!
Outputed cleaned crime data done!
Merged crime and weather data done!
Outputed combined data done!
>>> wdf = cd.weather_df
>>> wdf.head(10)
        Date  Temp  Prcp  Snow
0 2017-01-01  31.5  0.00   0.0
1 2017-01-02  36.0  0.24   0.0
2 2017-01-03  31.5  0.02   0.0
3 2017-01-04  15.5  0.00   0.0
4 2017-01-05  10.0  0.00   0.0
5 2017-01-06   5.5  0.00   0.0
6 2017-01-07  10.5  0.00   0.0
7 2017-01-08  12.0  0.00   0.0
8 2017-01-09  27.0  0.02   0.2
9 2017-01-10  39.5  0.20   0.0
```

Figure 7 Display the first 10 tuples of data frame ‘weather_df’

4.2.2 Crime data cleaning

Similarly, after importing crime data, attributes ‘Date’, ‘Community Area’, ‘FBI Code’, ‘Domestic’ and ‘Arrest’ are selected. As the crime items are not properly ordered, the object ‘sort_values’ is used to sort the tuples by date. To adjust the Date format to be the same with that in weather date, the time of the crime occurrence is removed. After running the script in Figure 8, we can obtain the data frame as shown in Figure 9.

```
# Read crime data
crime_df = pd.read_csv( "../data/raw/crime.csv" , sep = "," , parse_dates = ['Date'] )
print( 'Read crime data done!')

# Select attributes: Date, Community Area, FBI Code, Domestic, Arrest
crime_df = crime_df[ [ 'Date' , 'Community Area' , 'FBI Code' , 'Domestic', 'Arrest' ] ]
print( 'Select crime attributes done!')

# Sort by date
crime_df[ 'Date' ] = pd.to_datetime( crime_df.Date )
crime_df = crime_df.sort_values('Date')
print( 'Sorted crime date by date done!')

# Remove time, keep only date
temp = pd.DatetimeIndex( crime_df[ 'Date' ] )
crime_df[ 'Date' ] = temp.date
crime_df[ 'Date' ] = pd.to_datetime( crime_df.Date )
print( 'Removed time from datetime done!' )

# Output cleaned crime data to csv file
crime_df.to_csv( '../data/cleaned/cleaned_crime.csv' )
print( 'Outputed cleaned crime data done!')
```

Figure 8 Python script for cleaning crime data

	Date	Community Area	FBI Code	Domestic	Arrest
213648	2017-01-01	44	14	False	False
253112	2017-01-01	68	02	False	False
239966	2017-01-01	29	20	True	False
239972	2017-01-01	24	11	False	False
240360	2017-01-01	26	02	True	False
259904	2017-01-01	32	02	False	False
240649	2017-01-01	67	02	False	False
240922	2017-01-01	25	02	True	False
259889	2017-01-01	29	20	False	False
137179	2017-01-01	73	11	False	False

Figure 9 Display the first 10 tuples of data frame ‘weather_df’

4.2.3 Combination of cleaned weather and crime data

Based on the data generated by 1.1 and 1.2, we can simply combine them into a new dataset by the object ‘merge’. The cleaned weather and crime data share the same column ‘Date’, which is applied as the field name to join on. The python script and generated output demo are shown in Figure 10 and Figure 11 respectively.

```
# Merge crime and weather data frames
combine_df = pd.merge( weather_df , crime_df , how = 'outer' , on = 'Date' )
print( 'Merged crime and weather data done!')

# Output merged data to csv file
combine_df.to_csv( '../data/cleaned/combined.csv' )
print( 'Outputed combined data done!')
```

Figure 10 Python script for merging cleaned weather and crime data

	Date	Temp	Prcp	Snow	Community Area	FBI Code	Domestic	Arrest
0	2017-01-01	31.5	0.0	0.0	44	14	False	False
1	2017-01-01	31.5	0.0	0.0	68	02	False	False
2	2017-01-01	31.5	0.0	0.0	29	20	True	False
3	2017-01-01	31.5	0.0	0.0	24	11	False	False
4	2017-01-01	31.5	0.0	0.0	26	02	True	False
5	2017-01-01	31.5	0.0	0.0	32	02	False	False
6	2017-01-01	31.5	0.0	0.0	67	02	False	False
7	2017-01-01	31.5	0.0	0.0	25	02	True	False
8	2017-01-01	31.5	0.0	0.0	29	20	False	False
9	2017-01-01	31.5	0.0	0.0	73	11	False	False

Figure 11 Display the first 10 tuples of merged data frame 'combine_df'

4.2.4 Formalizing data for DMak

Previous procedures extract all useful features from raw data. However, to use it in DMak software, we have to convert the combined data set to the format of DMak input data, as shown in Figure 12.

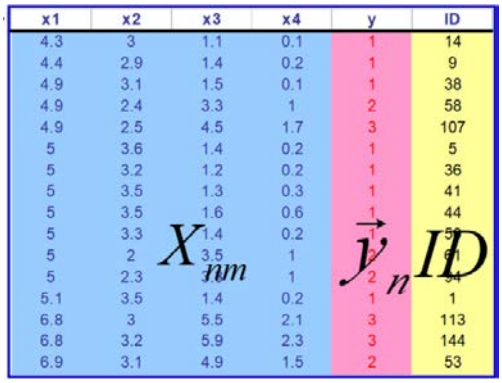

Figure 12 MetaNeural format

Several additional steps need to be completed to formalize the combined data: remove 'Date' column, replace FBI code by sequenced number (*see Appendix B for the details*), and replace Boolean 'Domestic' and 'Arrest' by integers ([True, false] → [1, 2]). By executing python script 'dmak.py', the generated data demo is shown in Figure 13.

```
>>> dmak = pd.read_csv('../data/cleaned/dmak.csv')
>>> dmak.head(10)
  Temp\tPrcp\tSnow\tCommunity Area\tFBI Code\tDomestic\tArrest\tID
0                    31.5\t0.0\t0.0\t44\t1\t2\t2\t1
1                    31.5\t0.0\t0.0\t68\t2\t2\t2\t2
2                    31.5\t0.0\t0.0\t29\t3\t1\t2\t3
3                    31.5\t0.0\t0.0\t24\t4\t2\t2\t4
4                    31.5\t0.0\t0.0\t26\t2\t1\t2\t5
5                    31.5\t0.0\t0.0\t32\t2\t2\t2\t6
6                    31.5\t0.0\t0.0\t67\t2\t2\t2\t7
7                    31.5\t0.0\t0.0\t25\t2\t1\t2\t8
8                    31.5\t0.0\t0.0\t29\t3\t2\t2\t9
9                   31.5\t0.0\t0.0\t73\t4\t2\t2\t10
```

Figure 13 data with MetaNeural format

Fortunately, we didn't find any missing or outliers in our datasets.

4.2.5 Balance dataset

Among 266635 cases, only around 20% (51590) of the criminals are arrested, which means the data is strongly unbalanced. Utilizing function "resample" in sklearn.utils, the majority class (not arrested) can be down sampled to be the same size of minority class. Then the down sampled class and the minority class are combined and shuffled. We use this new dataset as input of DMak software.

5. Results of paralleled linear regression

5.1 Data-oriented workflow

As Figure 14 shows, the collected weather and crime data are imported into python script "clean_data.py". This script retained only useful features. Both datasets have the feature "Date", which is applied as the field name to join on. The merged dataset is outputted to file "combined.csv", as the input of "dmak.py".

"dmak.py" is used to convert data format of new dataset into MetaNeural. The major modifications conducted in this script include removing 'Date' column, replacing FBI code by sequenced number and replacing Boolean 'Domestic' and 'Arrest' by integers ([True, false] to [1, 2]).

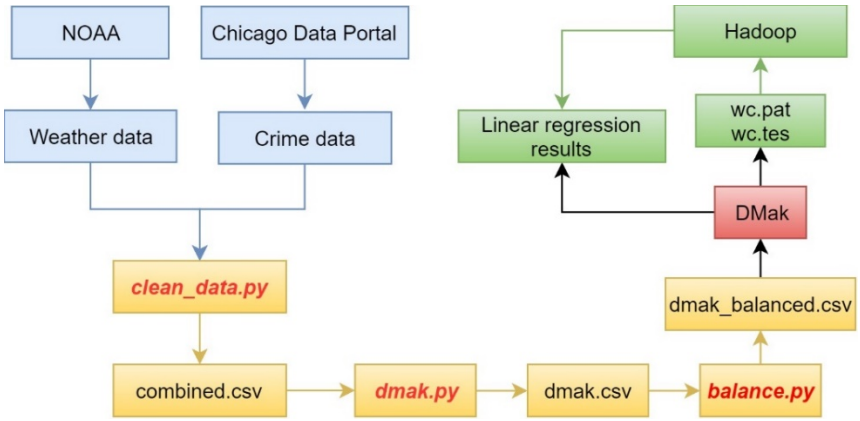

Figure 14 Data workflow of performing linear regression on weather-crime dataset

Figure 15 describe the data workflow in Hadoop, which is a detailed description of green part in Figure 14. Script "training_mapper.py" uses training data "wc.pat" as input. The mapper-generated key-value pairs are sorted and imported into script "training_reducer.py". It computes the linear regression coefficients (weights) and outputs the results into file "weights.txt", which, together with test data "wc.tes", are fed to script "test_mapper.py". "test_mapper.py" generates the predictions and targets, which are the key-value pairs of test mapper. It also writes the averages of predictions and targets to file "tes-ave.txt". Script "test_reducer.py" uses the values of predictions and targets and their averages to compute performance metrics of linear regression, which are $q^2, Q^2, MAE, and\ MSE$.

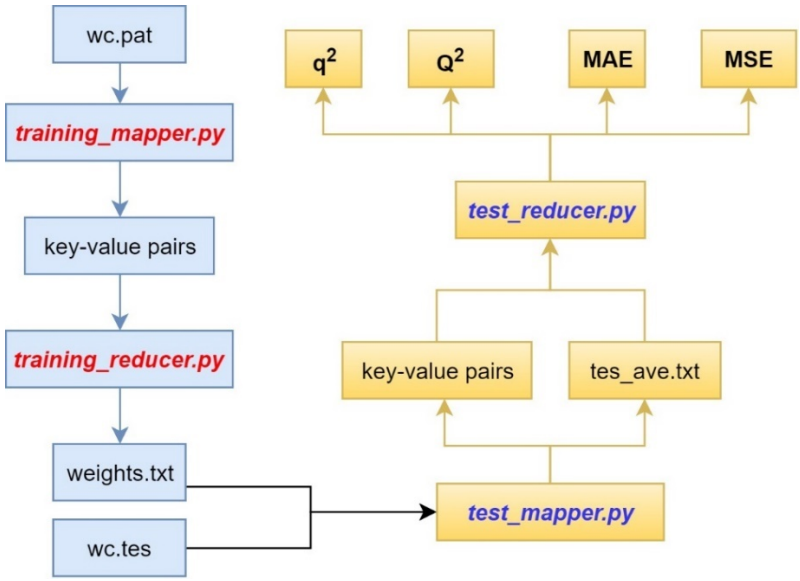

Figure 15 Data workflow in Hadoop MapReduce

5.2 Results

Following the data workflow in previous section, we performed our linear regression model on both Boston Housing dataset and our weather-crime dataset. The comparison of linear regression weights, and error metrics (q^2, Q^2, MAE, and MSE) with DMak and WEKA software are listed in below four tables, which indicates that for both datasets, our MapReduce model produces exactly same weights, and error metrics (q^2, Q^2, MAE, and MSE) with DMak software. This proves that our MapReduce linear regression model is correct.

Table 1 Boston Housing dataset - Linear regression coefficients (weights)

Attributes	Hadoop (Scaled)	Hadoop (Descaled)	DMAK	WEKA
CRIM	-0.09406	-1.0057034E-01	-0.10057	-0.0996
ZN	0.128322	5.0603387E-02	0.0506	0.0501
INDUS	-0.00578	-7.7492255E-03	0.00775	-
CHAS	0.1018	3.6861522E+00	3.68615	3.6311
NOX	-0.23021	-1.8271809E+01	-18.2718	-18.0028
RM	0.244139	3.1957311E+00	3.19573	3.2314
AGE	0.018228	5.9556486E-03	0.00596	-
DIS	-0.34249	-1.4958855E+00	-1.49589	-1.5169
RAD	0.27858	2.9425254E-01	0.29425	0.2937
TAX	-0.2459	-1.3418653E-02	-0.01342	-0.0136
PTRATIO	-0.22964	-9.7554921E-01	-0.9756	-0.9732
B	0.071596	7.2126654E-03	0.00721	0.0072
LSTAT	-0.39448	-5.0805656E-01	-0.5081	-0.5013
CONST	-0.09406	4.2000582E+01	42.0006	41.7686

Table 2 Boston Housing dataset - Performance metrics comparison

Metrics	q2	Q2	MAE	MSE
Hadoop	0.28426	0.29503	4.01688	5.71589

DMak	0.2843	0.2950	4.017	5.716
WEKA	-	-	4.0094	5.7342

Table 3 Weather-crime data Linear regression coefficients (weights)

Attributes	Hadoop (Scaled)	Hadoop (Descaled)	DMAK (Descaled)
Temp	0.000536	1.4541303E-05	0.00001
Prcp	-0.0015	-2.0655521E-03	0.00207
Snow	0.00187	2.9920623E-03	0.00299
Community	-0.04025	-9.5046146E-04	0.00095
FBI-code	-0.30155	-2.8375660E-02	0.02838
Domestic	0.029593	4.1027922E-02	0.041031
CONST	-	1.7182813E+00	1.71828

Table 4 Weather-crime data Performance metrics compariso

Metrics	q2	Q2	MAE	MSE
Hadoop	0.90852	0.90853	0.4547	0.4766
DMak	0.9085	0.9085	0.455	0.477

Reference

1. Hadoop, A. *Apache Hadoop Releases*. Available from: http://hadoop.apache.org/releases.html.
2. Hong, K. *HADOOP 2.6 INSTALLING ON UBUNTU 14.04 (SINGLE-NODE CLUSTER)* Available from: http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php.
3. DeRoos, D., et al., *Hadoop for dummies*. 2014: John Wiley & Sons, Incorporated.
4. Hong, K. *HADOOP 2.6 RUNNING A MAPREDUCE JOB (SINGLE-NODE CLUSTER)*. Available from: http://www.bogotobogo.com/Hadoop/BigData_hadoop_Running_MapReduce_Job.php .
5. Radtka, Z. and D. Miner, *Hadoop with Python*. First ed. 2016.

6. Noll, M. *Writing An Hadoop MapReduce Program In Python*. Available from: http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/#prerequisites.
7. NOAA. *Climate Data Online Search*. Available from: https://www.ncdc.noaa.gov/cdo-web/search.
8. DATA.GOV. *Crimes - One year prior to present*. Available from: https://catalog.data.gov/dataset/crimes-one-year-prior-to-present-e171f#sec-dates.