Machine Learning Application on Spark - Sign Language Recognition

YU HOU, The University of California, USA
ZUER WANG, The University of California, USA
YUHAN SHAO, The University of California, USA

We built a distributed sign-language recognition systems on the mobile Spark platform. To implement the hand sign-language, we used ASL Alphabet as the database, which is an image data set for alphabets in American Sign Language. We used Inverse Kinematics method to convert Cartesian coordinates to the rotation angles. Using these rotation angles as the training data, we ran Naive Bayes model in the PySpark environment. In order to emulate the mobile environment and validate the feasibility of running Spark applications on mobile devices, we configured different network settings and discovered the effects of each traffic condition on the Jobś execution time.

ACM Reference Format:
Yu Hou, Zuer Wang, and Yuhan Shao. 2023. Machine Learning Application on Spark - Sign Language Recognition . 1, 1 (August 2023), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The project aims to implement a machine learning related application which could run on the mobile Spark environment. We use MediaPipe for image processing and use Naive Bayesian network for hand-sign language recognition with the

2 BACKGROUND

2.1 Naive Bayes

Naive Bayes is an estimator and a basic classification algorithm. The premise of this algorithm is that all the features are independent of each other. Naive Bayes training is very efficient since after one-time traverse of the training data, it calculates the conditional probability of each feature for a given label, and then applies Bayes theorem to calculate the conditional probability of the label given a data sample, and finally uses this trained conditional probability to predict.

Machine Learning Library (MLlib) supports multinomial Naive Bayes, which is often used for document classification. Multinomial Naive Bayes classifier is appropriate for classification with discrete features, for example, the number of occurrences of a word. It takes RDD[LabeledPoint] and the smoothing parameter lambda as the inputs and outputs a Naive Bayes model that can be used for evaluation and prediction.

Authors' addresses: Yu Hou, The University of California, Los Angeles, USA, yuhou316@g.ucla.edu; Zuer Wang, The University of California, Los Angeles, USA, WangZuerVictoria@gmail.com; Yuhan Shao, The University of California, Los Angeles, USA, yuhan17@g.ucla.edu.

XXXX-XXXX/2023/8-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.2 Spark

Spark is a big data parallel computing framework based on memory computing. It improves the real-time performance of data processing in the big data environment, while ensuring high fault tolerance and high scalability and allowing users to deploy Spark on a large number of cheap hardware. Spark is an alternative to MapReduce and is compatible with distributed storage layers such as HDFS and Hive. Spark can be integrated into Hadoop ecosystem to make up for the deficiencies of MapReduce.

Spark partitions data in a distributed environment, then it converts multiple jobs into directed acyclic graphs (DAGs) and performs DAG scheduling and distributed parallel processing of tasks in stages. Spark architecture adopts the Master-Slave model in distributed computing, as shown in Fig. 1. Master is the node containing the Master process in the corresponding cluster, and the Slave is the node containing the Worker process in the cluster. Master, the controller of the entire cluster, is responsible for the normal operation of the entire cluster, while the Worker, the computing node, receives commands from the master node and reports status. The Executor is responsible for task execution, and the Client is responsible for submitting applications, and the Driver is responsible for controlling the execution of an application.

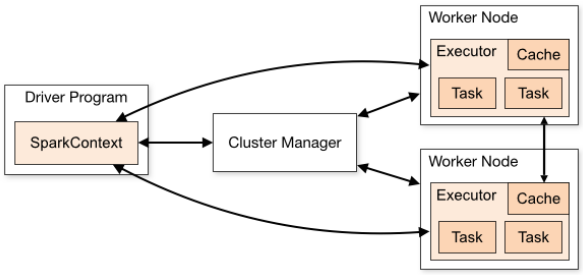


Fig. 1. Spark Cluster Overview

3 RELATED WORK

3.1 Naive Bayes

3.2 Spark

4 IMPLEMENTATION

4.1 Image Preprocessing

The first step of implementation is image preprocessing. We used the ASL Alphabet dataset, which is an image dataset for alphabets in American Sign Language. This dataset includes 26 classes, from A to Z, which represents 26 different hand gestures meaning. Since the original data consists of 21 Cartesian coordinates (x, y, z) of hand

landmarks for each right hand image, as shown in Fig. 2, we used Inverse Kinematics method to convert these Cartesian coordinates to the rotation angles of x, y, z-axis.

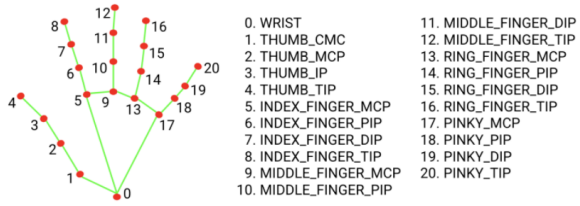


Fig. 2. 21 Cartesian coordinates in ASL Alphabet Dataset

4.2 Running Model in Spark Environment

The second step of implementation is running the model in the Spark environment. After we got the rotation angles of x, y, z-axis, we used Naive Bayes model to train the data, and the accuracy is 92.857%. Then, we ran the same model in the cluster Spark environment. First, we started a standalone master server by using the SparkConf static methods to create a SparkContext. Once started, the master would print out a URL for itself, which can be used to connect workers to it. Then, we started one or more workers and connected them to the master, with the personalized number of CPUs and memory. In this Spark environment, the accuracy is also 92.857%, which is the same as above.

4.3 Emulating the Mobile Environment

The third step of implementation is testing different combinations of worker-core in different network environments, in order to emulate the mobile environment and validate the feasibility of running Spark applications on mobile devices. To change the number of workers and cores, we set system properties by changing the spark-defaults configuration to set the executor memory and worker memory, and then we edited the configure Spark for our site to run various Spark programs in different numbers of workers and different numbers of cores. Specifically, we ran different combinations of worker-core in the default network environment. Since our laptop has 8 cores, (number of workers * number of cores for each worker) cannot be greater than 8, we ran the model with 1 worker-1 core, 1 worker-2 cores, 1 worker-4 cores, 1 worker-8 cores, 2 workers-1 core, 2 workers-2 cores, 2 workers-4 cores, 4 workers-1 core, 4 workers-2 cores, 8 workers-1 core. After running in different network settings, the optimal combination of worker-core is 1 worker-4 cores.

4.4 Analyzing the influence of Different Traffic Conditions

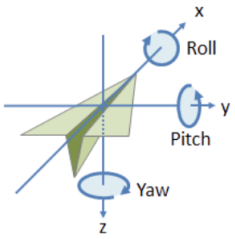
The fourth step of implementation is analyzing the influence of these 3 major variables that decide the speed of the network traffic: bandwidth, package dropped rate, and delay. In order to do this, we used the control variable method to find the relationships between execution time and these 3 variables respectively. Using the 1 worker-4 cores combination, we configured each variable respectively and found the effects of each variable to the network traffic.

5 ISSUE

During the implementation, we encountered many issues.

5.1 Rotation Angles

The first one is that when we convert Cartesian coordinates to the rotation angles3, we found some of the rotation angles around y and z-axis does not make sense, and the reason is the second knuckle and the third knuckle of the hand could only rotate around the x-axis, so we should not calculate the rotation angle using the y and z coordinate change.



Axis of Rotation	Euler Angle Name	Euler Angle Symbol
x	Roll	u
y	Pitch	v
z	Yaw	w

Fig. 3. Rotation Angles

5.2 Connection Between Master and Workers

The second one is that when we ran the model in the Spark environment, the terminal shows the error "Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources". The reason of it is Spark Master doesn't have any resources allocated to execute the Job like worker node or slave node, so what we did is to start the slave node by connecting with the master node. The conclusion we get is to start both the master node and slave node during spark-submit so that we will get enough resources allocated to execute the job.

5.3 Disable Debug and Info Logging

The third issue is when we ran the model in the Spark environment, the program was stuck at an error "added broadcast_0_piece0 in memory". The reason for this is we used the show function of the data frame from the SparkSession builder package to display part of the data frame. So, we used the history server method to check how the program runs inside the Spark and find that the show function costs a lot of time and makes the program crash. The conclusion from this is that although debug or info messages is useful if the dataset is small to display the data frame, it would result I/O operations and cause performance issues when running Spark jobs with greater workloads, so disabling debug and info logging could prevent the program crash if the dataset is large.

5.4 Network Simulation Issue

At the beginning, when simulate different network conditions (i.e.: packet loss and delay), notice that all the outputs are not same as what we expected. We find that it is because we use Mac xcode extension package – network link conditioner to adjust the network condition, which actually does not affect the local network. To fix

this problem, we use the command line4 to simulate the network environment. We learned that before doing an expensive test, we need to verify the correctness of our approach.

```
sudo pfctl -E
sudo dnctl pipe 1 config bw 10Mbit/s delay 5 plr 0.005
echo "dummynet out from any to any pipe 1" | sudo pfctl -f -
echo "dummynet in from any to any pipe 1" | sudo pfctl -f -
```

Fig. 4. Command line for network environment simulation

6 EVALUATION

6.1 Terminology

Workers Jobs Tasks (core)

6.2 Different Combinations of Workers and Cores

(worker:10GB, executor memory:10GB)	1 core	2 cores	4 cores	8 cores
1 Worker	6/0.8/0.2/0.6	5/0.6/0.2/0.4	5/0.6/0.2/0.5	5/0.7/0.3/0.7
2 Workers	6/1/0.2/1	7/0.8/0.3/0.9	8/0.8/0.2/1.0	**NONE**
4 Workers	10/0.9/0.5/1	8/0.8/0.3/1	**NONE**	**NONE**
8 Workers	17/0.9/0.3/3	**NONE**	**NONE**	**NONE**

Fig. 5. Form of different combinations of workers and cores

6.3 1 worker 4 cores - Bandwidth

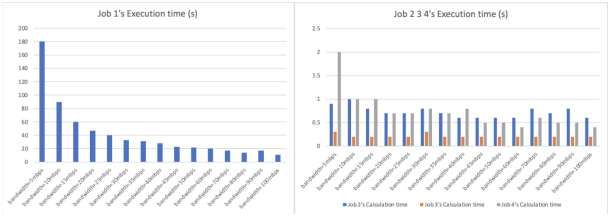


Fig. 6. 1 worker 4 cores - Bandwidth

6.4 1 worker 4 cores - Packets Dropped Percentage

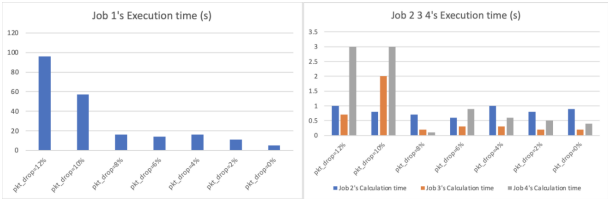


Fig. 7. 1 worker 4 cores - Packets Dropped Percentage

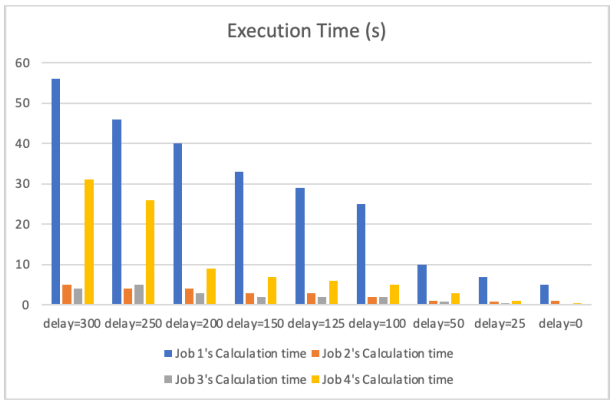


Fig. 8. 1 worker 4 cores - Delay

6.5 1 worker 4 cores - Delay

7 FUTURE WORK

7.1 Run on phones with different storage

Run with different storage – 2GB V.S. 10GB

7.2 Use serialized data format

In our current project, we use JSON file for serialized data. In order to do the Spark performance tuning, it would be better, if we can write an intermediate file in serialized and optimized formats like Avro, Kryo, Parquet, etc. Since we know that one Spark job writes data into a File and another Spark jobs read the data, process it, and writes to another file for another Spark job to pick up.

7.3 Use multiple computers

7.4 Extend to large dataset????

8 CONCLUSION