Sanyam Agrawal SE21UCSE192 CSE3

Report - Big Data (Minor Project)

Report on Hadoop-Based SVM Implementation

Introduction

The code implements Support Vector Machine (SVM) using Apache Hadoop's MapReduce framework. It's designed to handle large datasets distributed across a cluster. Here's a detailed review including functionality and key components of the code.

Overview of Code Structure

The code is structured into three main parts:

1) **SVMMapper Class (Mapper):**
- This class extends the Mapper interface from the Hadoop MapReduce framework.
- Its map method is responsible for processing the input data. It extracts features and labels from each input line (assumed to be space-separated) and emits intermediate key-value pairs. The keys are either individual feature indices ("feature" + i) with their corresponding values, or a "label" key associated with its value.

2) **SVMReducer Class (Reducer):**
- This class extends the Reducer interface from the Hadoop MapReduce framework.
- Its reduce method aggregates the values mapped to the same keys and applies the Sequential Minimal Optimization (SMO) algorithm to learn the SVM weights and bias.
- The learning rate, maximum iterations, and tolerance for the learning process can be adjusted through the job configuration.

3) **SVM Class (Driver):**
- This class contains the main method, which serves as the entry point of the application.
- It sets up the job's configuration, including the input and output paths, the SVM parameter C, and initializes and starts the MapReduce job.

Detailed Analysis of Key Components

1) **Input and Output Handling:**
- Input and output paths are set through command-line arguments which are passed to the FileInputFormat and FileOutputFormat respectively.
- The input data is considered to be a text format where each line contains space-separated numeric values, the last of which is the label.

2) **Mapper Logic and Data Parsing:**
- The map method of the SVMMapper class splits each input line into tokens using the split("\\s+") method.
- The last token is treated as the label, and the rest are considered as features.
- Key-value pairs are emitted for each feature as well as the label, using the context.write method.

3) **Reducer Logic and SVM Training:**
- The reduce method of the SVMReducer class differentiates between feature and label processing.
- For the "label" key, it aggregates all labels into a list for use in the SVM training process.
- Applying the SMO algorithm, it adjusts the weights and the bias based on the classification error and updates the conditions accordingly.

4) **Job Configuration and Execution:**
- The SVM parameter C, crucial for the regularization strength of the SVM, is configurable via job configuration.
- Controls job setup, including specifying the Mapper and Reducer classes, output key and value classes, and job name.

5) **Error Handling and Edge Cases:**
- Basic command-line argument validation is performed.
- Outputs to the console if the arguments do not match the expected scheme, helping in debugging and ensuring correct user input.

Workflow

The workflow of the Hadoop-based SVM implementation involves several stages orchestrated by the MapReduce framework, encompassing both data processing and SVM training, with the assistance of the Job Tracker and Task Tracker.

Initially, data ingestion begins with the input data stored in a distributed file system, where each line contains space-separated feature values and a corresponding label. This data is provided as input to the MapReduce job through the Job Tracker.

In the Mapper stage, the SVMMapper class parses each input line, extracting features and labels. For each feature, a key-value pair is emitted with the feature index as the key and the feature value as the value. Additionally, a key-value pair with the "label" key and its associated value is emitted. This process is facilitated by the Task Tracker, which oversees task execution and resource allocation across the cluster.

Following the Mapper stage, the MapReduce framework automatically shuffles and sorts the intermediate key-value pairs, ensuring that all values associated with the same key are grouped together. This intermediate data is then passed to the Reducer stage for further processing.

In the Reducer stage, the SVMReducer class receives the grouped key-value pairs, distinguishing between feature indices and the "label" key. For the "label" key, the reducer aggregates all label values into a list, leveraging the Task Tracker to manage parallel processing of data across multiple reducers. Subsequently, the SMO algorithm is applied to learn the SVM model parameters, including weights and bias, using the aggregated labels and feature values.

Throughout the workflow, the Job Tracker monitors job progress, coordinates task execution, and ensures fault tolerance by reallocating tasks in case of failures. Finally, the learned SVM model parameters, emitted as key-value pairs by the reducer, are written to the specified output path with the assistance of the Task Tracker. This workflow, orchestrated by the MapReduce framework in collaboration with the Job Tracker and Task Tracker, enables efficient SVM training on large-scale datasets distributed across a Hadoop cluster.

Architecture

The architecture of the Hadoop-based SVM implementation consists of the following components:

1) **Input Data:**
   - The input data, assumed to be in a text format with space-separated feature values and a label in each line, is stored in a distributed file system (e.g., HDFS) or a cloud-based storage service (e.g., Amazon S3, Google Cloud Storage).

2) **MapReduce Framework:**
   - The implementation utilizes the Apache Hadoop MapReduce framework to distribute the data processing and SVM training across a cluster of machines.

3) **Mapper:**
   - The SVMMapper class extends the Mapper interface provided by the Hadoop MapReduce framework.
   - It processes the input data, extracting features and labels, and emits intermediate key-value pairs.

4) **Reducer:**
   - The SVMReducer class extends the Reducer interface provided by the Hadoop MapReduce framework.
   - It receives the grouped key-value pairs from the mappers, aggregates the labels, and applies the SMO algorithm to learn the SVM model parameters.

5) **Driver:**
   - The SVM class contains the main method, which serves as the entry point of the application.
   - It sets up the job configuration, including the input and output paths, the SVM parameter C, and starts the MapReduce job.

6) **Output Data:**
   - The learned SVM model parameters, including the weights and bias, are written to the specified output path in the distributed file system or cloud-based storage.
   - The overall architecture leverages the scalability and fault-tolerance capabilities of the Hadoop MapReduce framework to enable efficient SVM training on large-scale datasets distributed across a cluster of machines.

Computational Considerations

- **Scalability**: Leveraging Hadoop, the implementation handles data distributed across a cluster, thus suitable for large datasets.

- **Performance**: The iterative nature of the SMO algorithm and its implementation in the reduce phase may impact performance, considering the overhead of MapReduce. Optimization for convergence and tuning of parameters such as learning rate and tolerance is critical.

- **Fault Tolerance:** Inherits Hadoop's fault tolerance capabilities, able to handle node failures gracefully during computation.

Conclusion

In conclusion, the Hadoop-based SVM implementation presents a robust foundation for conducting SVM classification on extensive datasets. Leveraging the MapReduce framework, it demonstrates scalability and reliability crucial for handling large-scale data processing tasks. However, to harness its full potential in real-world scenarios, meticulous attention to performance tuning and algorithmic enhancements is imperative. By optimizing parameters and refining algorithms, this implementation can achieve greater efficiency and efficacy in practical applications of large-scale machine learning. Thus, while the framework lays a strong groundwork, continuous refinement and optimization are necessary to realize its utmost capabilities in addressing complex machine learning challenges at scale.

Commands to run MapReduce Code:

1) making a directory in HDFS by name 'sanyamagrawal' and putting the input dataset "Input_AIDS_Classification_50000.txt" in it.

```
hdoop@SANYAM:/home/sanyam$ hdfs dfs -mkdir /sanyamagrawal
hdoop@SANYAM:/home/sanyam$ hdfs dfs -chmod 777 /sanyamagrawal
hdoop@SANYAM:/home/sanyam$ hadoop fs -put /mnt/c/Users/pc/Downloads/Input_AIDS_Classification_50000.txt /sanyamagrawal/
```

2) compiling map reduce code:

```
hdoop@SANYAM:/home/sanyam$ javac -classpath "/home/hdoop/hadoop-3.3.6/share/hadoop/client/*" /mnt/c/Users/pc/Downloads/Sanyam/SVM.jav
a
```

3) Making jar file:

```
hdoop@SANYAM:/home/sanyam$ jar -cvf SVM.jar -C /mnt/c/Users/pc/Downloads/Sanyam/ .
added manifest
adding: SVM$DoubleWritableArray.class(in = 1127) (out= 715)(deflated 36%)
adding: SVM$SVMMapper.class(in = 1813) (out= 776)(deflated 57%)
adding: SVM$SVMReducer.class(in = 3171) (out= 1617)(deflated 49%)
adding: SVM.class(in = 1762) (out= 960)(deflated 45%)
adding: SVM.java(in = 6104) (out= 1659)(deflated 72%)
```

4) Compiles SVM.java with Hadoop and Java 1.8 dependencies, including rt.jar in the boot classpath.

```
hdoop@SANYAM:/home/sanyam$ javac -classpath "/home/hdoop/hadoop-3.3.6/share/hadoop/common/*:/home/hdoop/hadoop-3.3.6/share/hadoop/hdf
s/*:/home/hdoop/hadoop-3.3.6/share/hadoop/mapreduce/*:/home/hdoop/hadoop-3.3.6/share/hadoop/yarn/*:/home/hdoop/hadoop-3.3.6/share/had
oop/tools/lib/*:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/rt.jar" -source 1.8 -target 1.8 -Xbootclasspath/p:"/usr/lib/jvm/java-8-open
jdk-amd64/jre/lib/rt.jar" /mnt/c/Users/pc/Downloads/Sanyam/SVM.java
```

5) Executes a Hadoop job using SVM.jar, with input from /sanyam/Input_AIDS_Classification_50000.txt and output to /Output__AIDS_Classification_1000, using an additional argument 1.0.

```
hdoop@SANYAM:/home/sanyam$ hadoop jar /home/sanyam/hadoop-3.3.6/hadoop-3.3.6/sbin/SVM.jar SVM /sanyamagrawal/Input_AIDS_Classificatio
n_50000.txt /Output__AIDS_Classification_1000 1.0
2024-05-15 22:40:54,752 INFO mapreduce.Job:  map 100% reduce 100%
```