Big Data Parallel Programming Project Report

M'boungou Colombe

May 2021

1 Introduction

We want to learn to classify movie reviews according to the emotional polarity : positive or negative. This is a textbook binary classification problem that we will tackle in a supervised learning fashion. We have found a dataset of labeled movie reviews from IMDB. It contains 50,000 reviews split evenly into 25 000 training and as many testing samples. We will use a SVM model to solve this classification model for a good generalization error.

Figure 1: Sentiment Analysis

2 Data pre-processing

One common and simple approach in NLP sentiment analysis is to build feature vectors from the frequencies of a set of words.

First the samples are split into a train folder and a test folder and then split again between positive and negative samples. The first step was to write a python script to merge all samples into one single .csv file. Each line of this

data-file contains one review, one separator character and the corresponding label. The date file size is 79 MB. We randomly split the train data (70%) from the test data (30%).

Some HTML mark-up and punctuation characters like the dot have been removed using the "replace" functionality of text editors. It prevents them from be mistaken for words further in the process and probably it improves the model accuracy.

Then, we have to extract features from text. After being sure there no invalid or missing values, we get rid of numbers which are not helpful because their carry emotional meaning on their own.

To split each character string into a list of words or tokens we apply the "RegexTokenizer" transformer.

By using the "RemoveStopWords" transformer, we delete grammar words because they do not carry a positive or negative polarity on their own. The next step is to count the occurrence of each word using "CountVectorizer" in every review and produce vectors containing the frequencies of the most frequent words.

IDF(Inverse Document frequency) returns a feature vector of inverse document frequency of the most frequents words. Idf = log((m + 1) / (d(t) + 1)), where m is the total number of documents, reviews in our case, and d(t) is the number of documents that contain term t. This expresses the fact that words that are extremely frequent across the whole corpus may carry less importance than rare ones.

3 Spark Implementations

3.1 Using MLlib Pipeline

The pipeline contains the aforementioned pre-processing steps RegexTokenizer, RemoveStopWords, CountVectorizer, IDF ,StringIndexer and a SVM classifier. The input of the classifier is a (1500,1) vector for each sample.

4 Results

The model reaches a test accuracy of 87.3% which is not bad given that we simply count words regardless the interaction between them and that no hyperparameter tuning has been performed yet. The train accuracy is 88.4% which is not far from the test accuracy. Hence, the model is not over-fitting. The accuracy could be improved by cleaning further the feature vector further and also using a Random Forest model.

A MLIB pipeline, a default partition RDD and a custom partition RDD were implemented and executed on a single node and a 4-worker cluster using google cloud's resources.

The tables 1 and 2 respectively present the training and testing time of different implementations on different set-ups. It should be noted that the computing can vary for example from 30 to 45 seconds for the same implementation and cluster type.

Cluster	Single Node	Master + 4 workers
MLIB	45	50
RDD Default	543	308
RDD 2 partitions	419	290
RDD 4 partitions	413	282
RDD 8 partitions	414	285

Table 1: Testing time (in seconds) variation across different implementations and cluster types

Cluster	Single Node	Master + 4 workers
MLIB	0.29	0.23
RDD Default	0.31	0.35
RDD 2 partitions	0.32	0.53
RDD 4 partitions	0.23	0.35
RDD 8 partitions	0.20	0.30

Table 2: Testing time (in seconds) variation across different implementations and cluster types