Bayesian Inference Using Apache Spark

David Marcano*

Adviser: Taylor Arnold †

I. Introduction

In recent years the use of Bayesian inference for the analysis of statistical models has dramatically increased due to the proliferation of computing power. For most real-world applications, carrying out these Bayesian inferences turn out to be mostly numeric and computationally heavy since analytic forms of these models are mostly unfeasible. This is especially true about high-dimensional, non-linear models, which are mostly analyzed via Markov chain Monte Carlo (MCMC) techniques which are known for requiring notorious amounts of computation power. Thus, it is of interest to find ways to parallelize MCMC techniques in order to speed up Bayesian analysis.

Until recent years, no open-source, distributed computing, parallel framework existed. In 2011, Apache Hadoop revolutionized the way people and companies were storing, processing, and analyzing data. A distributed, parallel, and scalable software framework became the preferred tool of the large-data processing community. In 2014, a new framework called Apache Spark improved on the ideas of Hadoop. Spark made the data-processing step much faster than Hadoop by doing most work in-memory and improving on Hadoop’s MapReduce paradigm. This proposal will introduce Apache Spark, the Bayesian Framework, and Gibbs sampling. The goal of this project is to build a modular, domain-indefinite application programming interface (API) for Spark which will house all Bayesian modeling logic, including the parallelization of the well known MCMC technique known as Gibbs sampling. Both Bayesian inference and Gibbs sampling have not been implemented in Spark. I propose an implementation of Gibbs sampling in Spark, which in turn will be used for Bayesian inference.1

---

* Yale University, Double Major in Computer Science and Statistics
† Yale University Department of Statistics, Lecturer. AT&T Labs Research, Senior Scientist
II. Apache Spark

Apache Spark is an open-source, cluster computing framework that improves upon Hadoop. Spark requires cluster management (e.g. Hadoop YARN, Apache Mesos, or standalone), and a distributed storage system (e.g. Hadoop Distributed File System (HDFS), Cassandra, etc.). A huge advantage of Spark over Hadoop is its ability to hold and subsequently process the necessary data in-memory.

The primary abstraction in Spark is called Resilient Distributed Datasets (RDDs), a data structure partitioned across machines in the cluster. RDDs are created either by reading in external data and parallelizing the content, or by applying transformations on other RDDs.

Spark’s core language is Scala, a functional programming language that parallelizes really well. Scala is designed to compile to Java bytecode, thus all compiled Spark code runs on a Java virtual machine (JVM). A comfortable API in Java, Python, Scala, and R is available for Spark. Since the core language of Spark is Scala, this project will focus on building the API in Scala to forgo any overhead necessary in the serialization and deserialization of objects across languages.

The Spark architecture is as follows: A cluster is composed of one Driver and one or more Executors. A Spark process is coordinated by the SparkContext object in the driver program, which is the entry point for communication between the driver and the nodes. The underlying cluster manager is in charge of allocating resources across applications. Once the SparkContext recognizes the executors on the nodes, it sends the application code to the Executors, and every time data processing is required, the SparkContext sends tasks to the executors. The SparkContext is the abstraction for communicating with the scheduler (be it Mesos, YARN, or Spark’s own scheduler), and as long as the process can acquire executors, SparkContext does not care which scheduler is being used.

III. Bayesian Inference

Bayesian inference is a method by which we can update the probability of a hypothesis as evidence comes to light. For the work in this paper, we will be interested in inference for the parameters in a given model. For a vector of parameters $\theta$ and a probability density model $p(y|\theta)$, we can treat $\theta$ as uncertain and thus they can have a prior density $\pi(\theta)$, and we can apply Bayes Theorem to get the posterior (i.e updated) density:

$$\pi(\theta|y) = \frac{p(y|\theta)\pi(\theta)}{p(y)}$$

where $p(y|\theta)$ is termed the likelihood: the likelihood that we will see this observed data given the parameter vector, $\pi(\theta)$ is the prior probability: one’s previous estimate of the probability distribution of $\theta$, $p(y)$ is the marginal probability of the observed data, and $\pi(\theta|y)$ is the desired updated probability of the
parameters given that we saw the observed data.

With low-dimensional data it is not too hard to analytically calculate the posterior density, especially if we make assumptions of independence. The difficulty comes in when we start getting high-dimensional data in non-linear models. In these models, in the simple case, we can have a prior distribution for the parameters \( \pi(\theta) \), but the rest of the expression requires us to do high-dimensional integration. This can be no only analytically impossible, but could also be numerically really hard. Even in low-dimensions, an analytic solution can be impossible. Let \( p(y|\theta) \sim \mathcal{N}(\mu, \sigma^2) \), and \( \pi(\theta) \sim \text{Beta}(\alpha, \beta) \), then to calculate the normalizing constant (the denominator), we would need to do:

\[
\int p(y|\theta)\pi(\theta)
\]

which is the integral of a Normal and Beta distributions, which is intractable.

For a large multivariate model, if \( p(y|\theta) \sim \text{Beta} \), there is no good prior to pair with the likelihood. In both of these cases, a numerical approximation via MCMC would be better to obtain the posterior distribution.

### IV. Gibbs Sampling

Gibbs Sampling is an MCMC algorithm for getting observations from a multivariate probability distribution when direct sampling is hard. It is especially useful in Bayesian inference since more often than not the posterior distribution is unknown. The idea behind Gibbs sampling is that it is simpler to sample from a conditional distribution than the joint distribution. Suppose we want \( k \) samples of \( Y = (y_1, ..., y_n) \) from a joint distribution \( p(y_1, ..., y_n) \). Then the \( i^{th} \) sample is \( Y_i = (y_{i,1}, ..., y_{i,n}) \). Then the algorithm is as follows:

1. Begin with some \( Y_0 \)

2. To get the next sample at point \( i \) (i.e. the \( i + 1^{th} \) sample), we sample each variable \( y_{(i+1),j} \) from the distribution conditioned on all other variables. The other variables consist of the most recently updated form of the variables. Specifically, if we are at the \( j^{th} \) component, we update that component based on the following distribution: \( p(y_j|y_{(i+1),1}, ..., y_{(i+1),j-1}, y_{(i),j+1}, ..., y_{i,n}) \).

3. Repeat \( k \) times

**Deliverables**

1. Code for an API built on Spark to do Bayesian inference through Gibbs Sampling.

2. A final project report
References