What's Cooking?: Recipe Classification in the Hadoop Ecosystem

Project Proposal

Yashi Kamboj
Shubhangi Rakhonde
Zayd Hammoudeh

CS286 – Fall 2015

Table of Contents

1. Team Members 1
2. Introduction 1
3. Requisite Disclosure 1
4. Project Goal 2
5. Proposed Solution Overview 2
   - 5.1. Ingredients Preprocessor 3
   - 5.2. Classification Algorithms 4
     - 5.2.1. Naïve Bayes Classification 4
     - 5.2.2. K-Nearest Neighbors (KNN) Classification 4
     - 5.2.3. Ensemble Method 5
     - 5.2.4. Deployment to a Cloud Based MapReduce Service 6
6. Division of Responsibility 6

List of References 8

What's Cooking?
Recipe Classification in the Hadoop Ecosystem

1. Team Members

- Yashi Kamboj (yashi.kamboj@sjsu.edu)
- Shubhangi Rakhonde (shubhangi.rakhonde@gmail.com)
- Zayd Hammoudeh (zayd.hammoudeh@sjsu.edu)

2. Introduction

In September 2015, Yummly.com posted a dataset of recipes on the data science website Kaggle [2]. Each recipe record in the dataset consists of: a list of ingredients, a cuisine type, and an identification number. Records are formatted in JavaScript Object Notation (JSON) as shown in figure 1.


```
{
    "id": 24717,
    "cuisine": "indian",
    "ingredients": [
        "tumeric",
        "vegetable stock",
        "tomatoes",
        "garam masala",
        "naan",
        "red lentils",
        "red chili peppers",
        "onions",
        "spinach",
        "sweet potatoes"
    ]
},
```


Figure 1 – Example Record for a Recipe of Cuisine Type "Indian"

The Yummly training set is comprised of 39,774 recipes consisting of 6,714 different ingredients spread across 20 international cuisine types

3. Requisite Disclosure

One of our team members (Zayd Hammoudeh) is using this dataset in a different course (CS256 – Topics in Artificial Intelligence). One of SJSU's definitions of cheating is: "Submitting work previously graded in another course unless this has been approved by the course instructor or by departmental policy". To ensure compliance with this policy, we will develop an entirely new implementation of the project from scratch using only the tools we learned in this class. We may reuse some of the same machine learning algorithms (e.g. K-Nearest Neighbors, Naïve Bayes Classifier, etc.), but none of the code will be reused. Ideally, even the writing of this code for any overlapping portions of the project will be assigned to other team members. Rather, our primary goal is to develop a more efficient version of our existing architecture. If this is a concern, please let us know.

4. Project Goal

Develop an ensemble classifier for this dataset using Oozie and the Hadoop ecosystem. An existing non-Hadoop based version of this project exists. To ensure originality, we will not reuse or reference any of that code. Instead, one of our key goals is to reduce the run time of that algorithm as it presently takes many hours to execute.

5. Proposed Solution Overview

Figure 2 shows the current implementation plan for our project. Since the solution will need to be multistage, we plan to use Oozie to manage the individual subtasks. The complete list of tools we plan to use from the Hadoop Ecosystem include:

1. Oozie
2. Spark
3. MapReduce
4. MapReduce DistributedCache [3]
5. **Possible** – MapReduce Custom InputFormat – JSONInputFormat [4]
6. **Possible** – Cloud-based Deployment

A few members of the team also have experience with Elastic MapReduce from Amazon. If time permits, it is important to us that we deploy our solution to Amazon (or a similar public cloud provide) to verify its runtime performance compared to the existing single-core, singled-threaded version.

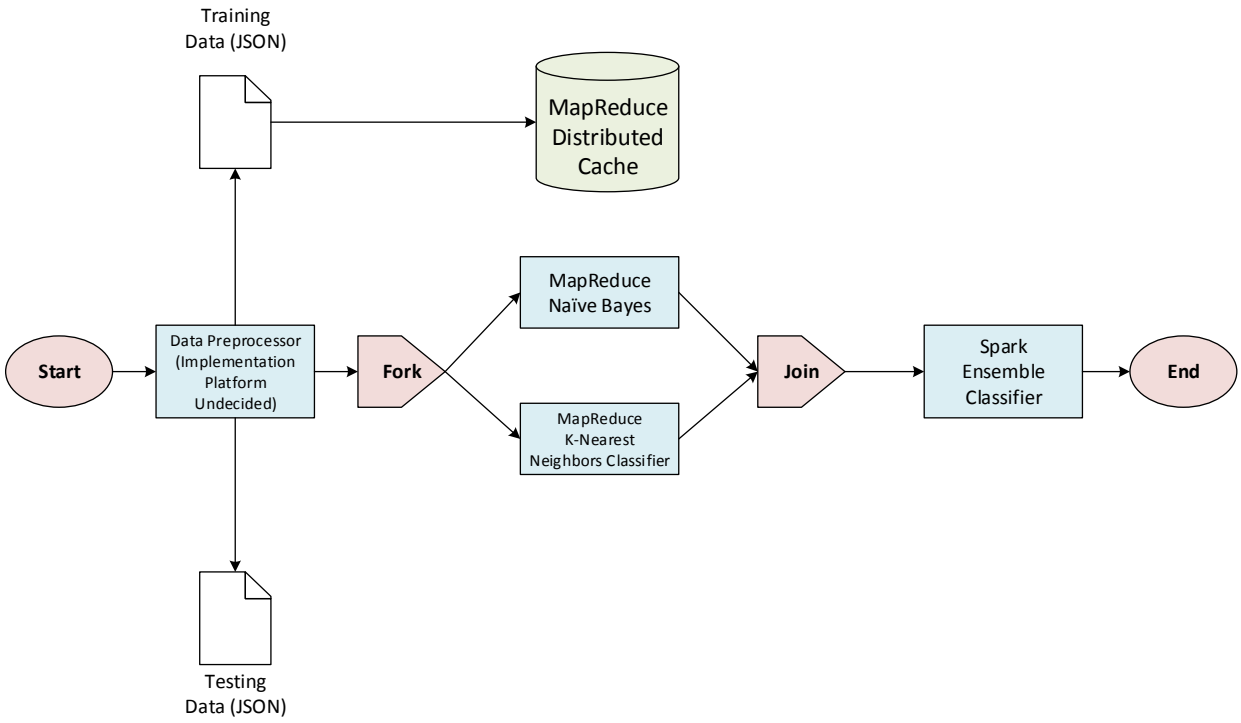

Figure 2 – Planned Software Architecture Design and Built Around our Oozie Workflow Model

5.1. Ingredients Preprocessor

In most scenarios, some degree of input data preprocessing is required for the machine learning algorithm to achieve the best possible results. This dataset is no exception. For instance, the ingredients list in each of the recipes is generated by different human beings. As with any natural language document, there is invariably some degree of variation where two people may use different phrases to refer to essentially the same object. An example that is relevant to this project would be where one person selects "low-fat sour cream" as recipe ingredient while a more health-conscious individual specifies the use of "non-fat sour cream"; A human can easily detect that both of these ingredients are for all intents and purposes the same. However, a computer which does only text or even substring comparison will mark these two ingredients as different. Hence, some degree of clean-up will be required on both the training and test sets to reduce the impact of this problem.

While the training set posted by Yummly to Kaggle is relatively small (i.e. less than 40,000 recipes), their entire recipe database is over 1 million recipes [1]. While we do not currently plan to use any advanced Natural Language Processing (NLP) techniques, we may want to exploit the scalability provided by Hadoop. Hence, we are considering different implementations for this preprocessor including Spark, standard MapReduce, Python, and Apache OpenNLP among others.

5.2. Classification Algorithms

For any classification problem, the most critical and difficult task is selecting the appropriate strategy for classification. The following subsections describe our current implementation plan (as shown in Figure 2). This is subject to change based off any unforeseen strategic requirements.

5.2.1. Naïve Bayes Classification

Unlike many other classification algorithms which require quantifiable attributes (e.g. Support Vector Machine, even k-Nearest Neighbor which we will discuss later), Naïve Bayes works as well (if not better) with nominal features. Hence, it is a good fit for this type of classification problem where we will be dealing with only binary features/attributes.

Depending on how easy it is to implement our problem using existing machine learning libraries (e.g. Mahout, Spark), we may decide to implement this portion using Hadoop specific tools. However, given the relative simplicity of Naïve Bayes and the time limitations of the project, we may simply implement this in standard MapReduce.

Using the Oozie "fork" construct, we can run Naïve Bayes in parallel with K-NN. While Naïve Bayes' execution is expected to be much faster than KNN (as explained in more detail in section 5.2.2), it nonetheless makes logical sense to have it controlled by Oozie both in terms of streamlining the execution as well as for data management.

5.2.2. K-Nearest Neighbors (KNN) Classification

K-Nearest Neighbor is a "lazy classifier" in that it does not require the development of a model in order to perform classification. Hence, for complex datasets where generating a model is exceptionally difficult, it can provide a useful tool for classification as it only relies on the premise that classification can be done based solely off the training records that are "most similar" to the unseen record [5].

Since KNN does not rely on a model to classify unseen records, a type of "time penalty" is paid when classifying such records. For a training set of size N and a testing/unseen record set of size M, the running time of the a K-Nearest Neighbor Algorithm is $O(N \cdot M)$; hence, its time complexity is far worse than many other classification techniques (e.g. Naïve Bayes, Random Forests, Support Vector Machine, etc.) that can be considered to run in constant time.

Although KNN has poor time complexity, the algorithm can be considered to be embarrassingly parallel. For example, consider again that there are N training records and you want to classify M unseen records. There are two possible techniques to parallelize this:

1. Pass the N training records to all P processing nodes. Also, pass disjoint sets of $\frac{M}{P}$ testing/unseen records to each of the P nodes. In the map stage, the processing nodes find the distance between each unseen record and all N training records. The reduce stage then uses each testing record's k nearest neighbors to determine that record's class value.

2. Pass a disjoint set of $\frac{N}{P}$ training records to each of the P processing nodes. Pass all M testing/unseen records to each of the P processing nodes. In the map stage, each processing nodes determine the k nearest neighbor for each unseen record given its set of training records. Then, for each unseen/testing record, the reduce stage selects the k nearest neighbors out of the $k \cdot P$ records passed to it and uses those k records to determine the class value.

It is our expectation that the runtime for the two approaches will be comparable and in the order of $O\left(\frac{N \cdot M}{P}\right)$. There may be some slight difference in overhead given the amount of data that must be sent over the network, but we expect this will contribute only a small portion to the overall runtime. If this assumption holds, we expect a near linear speed-up as more processing nodes are added. This can greatly reduce the runtime of our algorithm from a few hours to a much more manageable time.

Currently, we plan to use option KNN option #1 described above.

5.2.3. Ensemble Method

In sections 5.2.1 and 5.2.2, we discussed two different classification techniques. Each has its own strengths and weakness. Alone, each may have reasonable performance. However, when multiple classification algorithms are combined into a new composite algorithm, the resulting system may be more than a sum of its parts. This technique of combining different classification algorithms/model into a new model is known as the "ensemble method." [5]

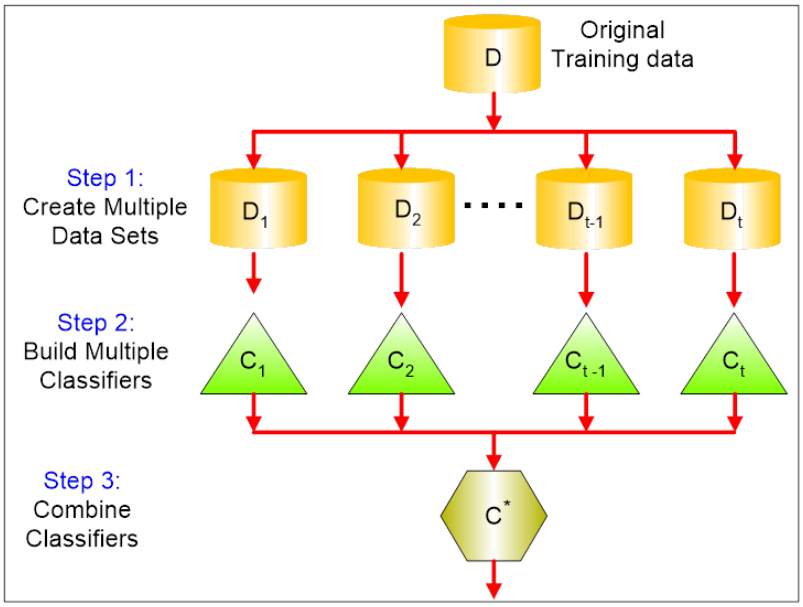

Figure 3 – Simple Example of an Ensemble Classifier [6]

Figure 3 shows the flow of a simple ensemble classifier. This example uses t different classifiers, each of which is supplied its own training dataset D_i (note the dataset for each classifier may be the same or different). In step #3, the results from the individual classifiers are synthesized into a single final result. It is important to note that our planned architecture (shown in figure 2) is intentionally similar to this model.

5.2.4. Deployment to a Cloud Based MapReduce Service

To verify the speed-up in performance versus an existing classifier, we plan to deploy our program to a cloud based MapReduce provider such as Amazon. This will allow us to exactly quantify the speed-up of this massively parallel approach taking into account all overhead (e.g. data transfer, Hadoop, etc.). If time does not permit us to do this, we will rely on calculations to quantify the speed-up, but getting this to work in the cloud is a key priority for us.

6. Division of Responsibility

The following is breakdown of what we see as the challenging/critical tasks for this project. Accompanying each task is a *tentative* task owner. This is subject to change based on the actual complexity of each of the tasks.

- Oozie Workflow Creation and Debug – **TBD**
- Distributed Cache Implementation– **TBD**
- Dataset Preprocessor – **TBD**

- Naïve Bayes Classifier – **TBD**
- K-Nearest Neighbors – **TBD**
- Ensemble Classifier – **TBD**
- Public Cloud Deployment – **TBD**

List of References

[1] "Introducing the Ultimate Cooking Tool," Yummly. [Online]. Available at: http://www.yummly.com/how-it-works/. [Accessed: Nov-2015].

[2] "What's Cooking?,". [Online]. Available at: https://www.kaggle.com/c/whats-cooking. [Accessed: Nov-2015].

[3] "DistributedCache (Apache Hadoop Main 2.7.1 API)." [Online]. Available at: https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/filecache/distributedcache.html. [Accessed: Nov-2015].

[4] "JsonInputFormat," pmr-common 2.0.1.0-0 API Documentation. [Online]. Available at: http://pivotal-field-engineering.github.io/pmr-common/pmr/apidocs/com/gopivotal/mapreduce/lib/input/jsoninputformat.html. [Accessed: Nov-2015].

[5] P. Tan and M. Steinbach, "Classification: Alternative Techniques," in *Introduction to Data Mining*, Boston: Pearson Addison Wesley, 2005, pp. 276–294.

[6] "Lecture Slides: Classification: Alternative Techniques," *Introduction to Data Mining*. [Online]. Available at: http://www-users.cs.umn.edu/~kumar/dmbook/dmslides/chap5_alternative_classification.ppt. [Accessed: Nov-2015].