CS294-1 Behavioral Data Mining Spring 2013

Programming Assignment 3: Distributed Machine Learning via Hadoop

Due: Monday April 15

The goal of this assignment is to process a moderately large dataset (a recent snapshot of Wikipedia) in distributed fashion using Hadoop. The data are available on the icluster, and accessible in the Hadoop File System. To be able to access hadoop, you need to set your PATH variable. Directions on how to do this, and some other background on running Hadoop on icluster, are available here:
http://inst.eecs.berkeley.edu/usr/pub/mapreduce.help
A script to set path variables for hadoop and BIDMat, are in ~cs294-1/adm/class.bash_profile. You can source this file (don't just execute it) from your .bash_profile file. If you still have the standard class account .bash_profile, just change the "MASTER" variable to cs294-1 and login again, and it should source the class profile. Once your path is set, you should be able to login to any icluster machine and do

```
hadoop  fs  –ls  /cs294_1/wikipedia/enwiki_100pct.xml
```

to see the file metadata. This is a single large text file (about 42 GB) containing all Wikipedia articles in XML format. There are smaller sample files: enwiki_10pct.xml and enwiki_1pct.xml which are 10% and 1% samples respectively, of the full dataset. You should work with these until your mapreduce pipeline is working efficiently. You can typically replace "-ls" with other unix filesystem commands "-mkdir", "-du" etc. Documentation of hadoop's HDFS commands is here:
http://hadoop.apache.org/docs/r0.18.1/hdfs_shell.html

You will train a classifier on the text content of Wikipedia articles to predict whether an article fits in a high-level category (ignoring its explicit category tag, which you will use for training only). Each article contains category information about the most specific categories it lies in, in a format like this:

[[Category:Graph Theory]]

These categories are generally too specific to be useful for classification. You should instead use a high-level category like, e.g. "computers" or "mathematics" (the exact choice is up to you). The links between categories and their subcategories are in the file: /cs294_1/wikipedia/subcats.txt in HDFS. The first step is to find all the subcategories below your target high-level category. You should use the subcats file to find all of its descendants by finding children from links in the subcats file, then children of children recursively etc, down to leaf nodes. The subcats.txt file contains entries like this:

(691898,'Fields_of_mathematics','GRAPH THEORY','2012-02-01 22:52:02','','uppercase','subcat')

The second field is the name of the parent category, the third is the "current" category. The first field (numeric) is a unique id for the current category. There may be more than one name (third field) for this category. Note that the formats of second and third fields are not the same. The second field contains no spaces (words linked by underscores) and has variable case. The third field has spaces and is all upper case. To find children of 'GRAPH THEORY' you would need to search for 'Graph_theory' in the second

field. Rather than trying to guess the case, it's better to convert both fields to a standard form, e.g. underscore-separated upper case.

Mapreduce Steps

Place your output and all intermediate results in a directory under the main Hadoop directory for the class, which is /cs294_1 in HDFS. You will need to write mapreduce jobs to do the following steps:

- Parse and tokenize the input. For this step you will probably want to use the specialized Lucene Wikipedia parser from here: http://lucene.apache.org/core/old_versioned_docs/versions/3_5_0/api/contrib-analyzers/index.html
- Create a dictionary for string tokens which maps each token to a unique ID. This dictionary should also contain token counts, and be sorted from most frequent to least frequent. You can (optionally) use bigram and trigram features by building dictionaries for those.
- Using the dictionaries, construct a numerical sparse matrix which is a bag-of-features representation of the document. e.g. element Mij should be the count of feature i in document j. Features will be individual words, bigrams or trigrams.
- Create a target vector which contains for each article a value of 1 or 0 depending on whether the article's category is a child of your main category.

Once the data is featurized, you will need to train your classifier. You can implement your classifier using either Logistic Regression or SVM. You cannot communicate between map tasks until the end of each mapreduce cycle, but you can update a model local to each mapper using stochastic gradient. At the end of the cycle, synchronize the models in the reduce step by averaging corresponding coefficients from all the mapper models.

You should hold out a random sample of data for cross-validation. E.g. take a random hash of every article's id number and designate certain values for training and the rest for testing. Ignore the test values during training, and ignore the training values during a test job.

Pragmatics

Hadoop jobs consume a lot of resources, and it's important to use them efficiently, especially in a shared environment. This is not a large dataset but still it will tax the modest icluster (20 nodes and 160 cores). Please follow the following guidelines:

- Write your code efficiently. Don't expect inefficient code to run in reasonable time on a cluster – there is considerable overhead built into Hadoop that offsets the number of machines working on your job.
- Run on the small data samples first. Start with the 1% sample. Assess the running time for each size job, and don't launch any job on the full-size dataset without determining how long it should run based on the smaller datasets.
- Use a reasonable number of mappers/reducers. The default should be fine, but since you're allowed to configure these numbers in the jobconf you may sometimes want to set the number smaller (usually) or larger (almost never) than the default. In the icluster configuration, there is a maximum of 160 tasks that can run concurrently, and the number of mappers/reducers should be smaller than this.

Some people think setting a larger number will make the job run faster. The opposite is true. If the system chooses a default much larger than this (thousands of mapper/reducers), then it is a sign that your job is bloated somehow (e.g. generating intermediate files that are much too large). Such jobs are likely to be killed as soon as we see them.

- None of your jobs should take more than a few hours to run. If we see jobs running longer than this, they are likely to be terminated.
- The input XML files will be split at arbitrary boundaries (for different mappers) by Hadoop's text file input routines. A few articles will be split across mappers and will not be processed because of this. Don't worry about it. But your code does need to handle these random start and end points within each mapper, and correctly process the articles that do not straddle a boundary.

Writeup

Please describe the sequence of steps of your Hadoop job, and the subsequent classification. There should be enough detail to allow someone else to reproduce the results. Include tables or graphs showing the performance of your classifier, including ROC and F1. Hand in a hardcopy and post a zip file with the code and graph data to the assignment 3 page in Bspace by 5pm Monday April 15.