libSVM and PSGD via MapReduce

hastagiri@inf.ethz.ch, lucic@inf.ethz.ch

April 17, 2013

1 libSVM

For the toy datasets that consists of 100k training examples and 20k test samples, you should train an SVM using the popular libSVM tool. Use the following steps and also play around with the options, especially with the class weights options to observe changes in overall prediction performance (accuracy). Note that libsvm expects datapoints in both test and training sets in the following format:

```
classLabel dimId:value dimId:value ...
```

We have already created an libsvm friendly format of the small dataset for you. Please proceed with the following steps.

1. Download libsvm as a tar/zip file from http://www.csie.ntu.edu.tw/ cjlin/libsvm/
2. Build using Make. You will see 2 new executables: svm-train and svm-predict.
3. Running svm-train with no options lists all the available options.
4. For our purposes use -t 0 to specify that you want to use a linear kernel. You can also play around with other options.
5. As the simplest case, you can train the SVM using

   ```
   ./svm-train -t 0 libsvm-train-small libsvm.model
   ```

   such that the output of the training phase is obtained in the file libsvm.model.
6. Now, one can test the performance on the file containing the test data using svm-predict. As an example, use

   ```
   ./svm-predict libsvm-test-small libsvm.model predict.out
   ```

   Apart from the overall accuracy on the test set, the individual predictions of the points in the test set are stored in predict.out

2 PSGD via MapReduce

You are provided with a set of files that will help you solve this task:

1. RealVector.java - defines basic operations with vectors.
2. TrainingInstance.java - represents one labeled feature vector.
3. SVM.java - represents a Support Vector Machine model. **You need to complete the method that trains a SVM given the training data set, learning rate and regularizer value.**

4. Performance.java - evaluator for the MapReduce solution. This is the same evaluator that is going to be used when you submit the solution.
5. PSGD.java - The class that implements Parallel Stochastic Gradient Descent. **You need to implement the Mapper and Reducer classes.**

Your workflow could be:

1. Implement/tweak PSGD.java and SVM.java.
2. Compile + pack as JAR.
3. Run with Hadoop to get the hyperplanes:

   ```
   hadoop jar PSGD.jar org.ethz.las.PSGD input output
   ```

4. Now get the hyperplanes in one file:

   ```
   hadoop fs -getmerge output_folder_from_last_step my.model
   ```

5. You can get the performance of the algorithm via:

   ```
   java -jar org.ethz.las.Performance my.model file_containing_training_instances
   ```

6. go to 1.