National College of Ireland

Supervised Unsupervised Learning in Spark Configuration Manual

MSc Research Project
Data Analytics

Vidya Sankar Velamuri
x15009653

School of Computing
National College of Ireland

Supervisor: Dr.Simon Caton

Supervised Unsupervised Learning in Spark Configuration Manual

Vidya Sankar Velamuri
x15009653
MSc Research Project in Data Analytics

12th September 2016

Abstract

Clustering is the unsupervised classification of patterns into groups and is one of the most popular techniques applied to explore and discover naturally occurring patterns within hitherto unlabelled data. The quality of the clusters resulting from a clustering algorithm can be verified using clustering validity indices, which take into account the intra cluster similarity and inter cluster separation of the clusters. However in a distributed setting the computation of pairwise distances between data points of a large data set distributed across the cluster can be computationally very expensive.

This research proposes to evaluate a sampling based approach to computing the cluster validity indices for distributed datasets and embed this methodology into a model selection pipeline that evaluates distributed machine learning jobs in selecting an optimal clustering algorithm. The results suggest the sampling error of the internal validation index so computed is statistically significant.

1 Configuration Manual

1. Shell Script : This shell script is used to install java on an bare ubuntu vm using VirtualBox. The steps download and install spark are well documented on the offical hadoop and spark project sites.

   (a) Shell Script to Install Java, Hadoop and Spark

```
# ubuntu

RUN apt-get update
RUN apt-get install -y curl wget tar sudo openssh-server
   openssh-client rsync
install -y curl wget tar sudo openssh-server openssh-client
   rsync

wget --no-cookies \
--no-check-certificate \
--header "Cookie: oraclelicense=accept-securebackup-cookie"
   \
"http://download.oracle.com/otn-pub/java/jdk/8u92-b14/jdk-8u92-linux-x64
   \
-O jdk-8-linux-x64.tar.gz

tar --strip-components=1 -xz jdk-8-linux-x64.tar.gz -C
   /usr/java/default/

sudo update-alternatives --install /usr/bin/java java
   /usr/java/default/jdk1.8.0_92/bin/java 1
sudo update-alternatives --install /usr/bin/javac javac
   /usr/java/default/jdk1.8.0_92/bin/javac 1
sudo update-alternatives --install /usr/bin/jar jar
   /usr/java/default/jdk1.8.0_92/bin/jar 1
```

   (b) bashrc :

```
export JAVA_HOME=/usr/java/default/jdk1.8.0_92
export PATH=$PATH:$JAVA_HOME/bin


export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:\$HADOOP_HOME/bin:$HADOOP_HOME/sbin

//create a symbolic link to the hadoop installation under
   /usr/local
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_COMMON_HOME=/usr/local/hadop
export HADOOP_HDFS_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/had
export HADOOP_YARN_HOME=/usr/local/hadoop
```

```
//create a symbolic link to the spark installation under
   /usr/local
export SPARK_HOME=/usr/local/spark2
export PATH=$SPARK_HOME/bin:$PATH
export SPARK_CONF_DIR=/etc/spark2


export HADOOP_CONF_DIR=/etc/hadoop
export YARN_CONF_DIR=/etc/hadoop

export HADOOP_LOG_DIR=/var/log/hadoop

//install maven to build hadoop and spark distributions and
   set up
export MAVEN_HOME=/opt/maven
export PATH=$PATH:$MAVEN_HOME/bin
```

(c) Hadoop Configuration : By default the conf files are under "HADOOP HOME/etc/hadoop" in the hadoop install folder. However for ease of copy files under to "/etc/hadoop" (Note this the etc folder under the root folder)

(d) hadoop-env.sh : Update JAVA HOME variable in this file

(e) slaves : this file represents the slaves where the yarn will start the DataNodes and Node Managers runs. The master node can also be listed here to start those daemons here

(f) core-site.xml

```
<configuration>

<!-- The Address of the name node -->
  <property>
    <name>fs.default.name</name>
    <value>hdfs://172.25.1.210:9000</value>
  </property>

  <property>
    <name>hadoop.tmp.dir</name>
    <value>/home/hduser/tmp/</value>
  </property>

</configuration>
```

(g) hdfs-site.xml

```
<?xml version="1.0"?>
<!-- hdfs-site.xml -->
<configuration>
  <!-- replication value for hdfs data -data replicated on
     2 nodes with value 2 -->
  <property>
    <name>dfs.replication</name>
```

```
    <value>2</value>
  </property>
  <!-- directory for the name node -->
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>/home/hduser/name</value>
  </property>
  <!-- data directory for the data nodes -->
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>/home/hduser/data</value>
  </property>
  <property>
    <name>dfs.namenode.checkpoint.dir</name>
    <value>/home/hduser/namesecondary</value>
  </property>
  <property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>172.25.1.210:9001</value>
  </property>
  <property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
  </property>
</configuration>
```

(h) mapred-site.xml

```
<!-- Put site-specific property overrides in this file. -->

<configuration>
   <!-- configuring yarn as the map reduce framework ? -->
   <property>
      <name>mapreduce.framework.name</name>
      <value>yarn</value>
   </property>
      <property>
         <name>mapreduce.jobhistory.address</name>
         <value>172.25.1.210:10020</value>
      </property>
      <!-- The Address for Map Reduce Job History Server -->
      <!-- This is also relevant for the Spark History
         Server and YARN log aggreation -->
      <property>
         <name>mapreduce.jobhistory.webapp.address</name>
         <value>172.25.1.210:19888</value>
      </property>
</configuration>
```

(i) yarn-site.xml

```
<configuration>
   <!-- Configuration for the Shuffle Service -->
   <!-- This is also relevant for Dynamic Resource
      Allocation -->
   <!-- relevant to the intermediate shuffles from the
      terminated executors ? -->
   <!-- refer to Running on YARN dynamic allocation for
      further notes -->
   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
   <property>
      <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
      <value>org.apache.hadoop.mapred.ShuffleHandler</value>
   </property>
   <property>
      <name>yarn.resourcemanager.address</name>
      <value>172.25.1.210:8032</value>
   </property>
   <property>
      <name>yarn.resourcemanager.scheduler.address</name>
      <value>172.25.1.210:8030</value>
   </property>
   <property>
      <name>yarn.resourcemanager.resource-tracker.address</name>
      <value>172.25.1.210:8031</value>
   </property>
   <property>
      <name>yarn.resourcemanager.admin.address</name>
      <value>172.25.1.210:8033</value>
   </property>
   <!-- The port on which the RM is exposed -->
   <property>
      <name>yarn.resourcemanager.webapp.address</name>
      <value>172.25.1.210:8088</value>
  <property>
      <name>yarn.nodemanager.resource.memory-mb</name>
      <value>4096</value>
   </property>
   <property>
      <name>yarn.nodemanager.resource.cpu-vcores</name>
      <value>3</value>
   </property>
<!--
   For the server node for which I had a m1.xlarge I had
      the following configuration
   <property>
      <name>yarn.nodemanager.resource.memory-mb</name>
      <value>8192</value>
   </property>
```

```
  <property>
     <name>yarn.nodemanager.resource.cpu-vcores</name>
     <value>4</value>
  </property>
-->
  <property>
     <name>yarn.application.classpath</name>
     <value>/usr/local/hadoop/etc/hadoop,
        /usr/local/hadoop/share/hadoop/common/*,
        /usr/local/hadoop/share/hadoop/common/lib/*,
        /usr/local/hadoop/share/hadoop/hdfs/*,
        /usr/local/hadoop/share/hadoop/hdfs/lib/*,
        /usr/local/hadoop/share/hadoop/mapreduce/*,
        /usr/local/hadoop/share/hadoop/mapreduce/lib/*,
        /usr/local/hadoop/share/hadoop/yarn/*,
        /usr/local/hadoop/share/hadoop/yarn/lib/*</value>
  </property>
  <property>
     <description>
      Number of seconds after an application finishes
         before the nodemanager's
      DeletionService will delete the application's
         localized file directory
      and log directory.

      To diagnose Yarn application problems, set this
         property's value large
      enough (for example, to 600 = 10 minutes) to permit
         examination of these
      directories. After changing the property's value,
         you must restart the
      nodemanager in order for it to have an effect.

      The roots of Yarn applications' work directories is
         configurable with
      the yarn.nodemanager.local-dirs property (see
         below), and the roots
      of the Yarn applications' log directories is
         configurable with the
      yarn.nodemanager.log-dirs property (see also below).
     </description>
     <name>yarn.nodemanager.delete.debug-delay-sec</name>
     <value>600</value>
  </property>

  <!-- Enable YARN Log Aggreation -->
  <property>
     <name>yarn.log-aggregation-enable</name>
     <value>true</value>
  </property>
  <property>
```

```
      <name>yarn.nodemanager.remote-app-log-dir</name>
      <value>/app-logs</value>
   </property>
   <property>
      <name>yarn.nodemanager.remote-app-log-dir-suffix</name>
      <value>logs</value>
   </property>
   <property>
      <name>yarn.log.server.url</name>
      <value>http://172.25.1.210:19888/jobhistory/logs</value>
   </property>

</configuration>
```

(a) Spark Configuration :

(b) spark-env.sh

```
export SPARK_CONF_DIR=/etc/spark
export HADOOP_CONF_DIR=/etc/hadoop

#add this line to export
   JAVA_HOME=/usr/java/default/jdk1.8.0_92 to
   /usr/local/spark/sbin/start-slave.sh

export JAVA_HOME=/usr/java/default/jdk1.8.0_92
export SPARK_MASTER_IP=172.25.1.210
export SPARK_WORKER_CORES=3
export SPARK_WORKER_INSTANCES=1
export SPARK_MASTER_PORT=7077
export SPARK_WORKER_MEMORY=4g
export
   MASTER=spark://${SPARK_MASTER_IP}:${SPARK_MASTER_PORT}


export SPARK_LOCAL_IP=172.25.1.222
export SPARK_PUBLIC_DNS=87.44.4.153
```

2. Start and Stop Hadoope

```
#To Install mllib to local maven repository
#=========================================
build/mvn -pl :spark-mllib_2.11 -Pyarn -Phadoop-2.6 clean
   install -DskipTests -Dmaven.javadoc.skip=true
   -Dmaven.test.skip=true


#Making a Dev Distribution for Spark Code
#=========================================
./dev/make-distribution.sh --name myspark --tgz -Phadoop-2.6
   -Phive -Phive-thriftserver -Pyarn -DskipTests
```

```
#Back up current build file to old
mv ~/build/spark-2.1.0-SNAPSHOT-bin-myspark.tgz
   ~/build/spark-2.1.0-SNAPSHOT-bin-myspark.tgz.old

#Move new build to build dir
mv ~/coderepos/spark/spark-2.1.0-SNAPSHOT-bin-myspark.tgz
   ~/build

#Unzip the tar file
tar -xvf ~/build/spark-2.1.0-SNAPSHOT-bin-myspark.tgz ~/build/

#Stop all Spark , YARN and HDFS services
$SPARK_HOME/sbin/stop-all.sh &&
   $HADOOP_COMMON_HOME/sbin/stop-yarn.sh &&
   $HADOOP_COMMON_HOME/sbin/stop-dfs.sh

#Remove Symbolic Links to installation
sudo rm /usr/local/spark2

#Remove the current installation
sudo rm -rf /usr/local/spark-2.1.0-SNAPSHOT-bin-myspark

#Move the new build to /usr/local
sudo mv ~/spark-2.1.0-SNAPSHOT-bin-myspark /usr/local/

#create a spark2 symbolic link again
sudo ln -s /usr/local/spark-2.1.0-SNAPSHOT-bin-myspark
   /usr/local/spark2

#fix issue with finding JAVA_HOME when launching Spark Worker
sudo cp ~/build/spark-class /usr/local/spark2/bin/spark-class

#restart HDFS, YARN and Spark
$HADOOP_COMMON_HOME/sbin/start-dfs.sh &&
   $HADOOP_COMMON_HOME/sbin/start-yarn.sh &&
   $SPARK_HOME/sbin/start-all.sh


#Make sure to enable log aggreation in YARN and configure both
   MapReduce Server and Spark History Server
#in the mapred-site.xml and spark_defaulf.conf respectively.

# Example:
# spark.master            spark://master:7077
#these values are important for enabling Spark History Server
#the directory are to be created before hand and use the name
   node and port on which launched
# references
   :http://spark.apache.org/docs/latest/monitoring.html
```

```
#
   https://www.ibm.com/support/knowledgecenter/SSGSMK_7.1.1/management_sym/
 spark.eventLog.enabled   true
 spark.eventLog.dir
    hdfs://172.25.1.210:9000/shared/spark-logs
 spark.history.fs.logDirectory
    hdfs://172.25.1.210:9000/shared/spark-logs
 spark.history.retainedApplications 50
 spark.history.fs.update.interval 10s
# spark.serializer
   org.apache.spark.serializer.KryoSerializer
# spark.driver.memory     5g
# spark.executor.extraJavaOptions -XX:+PrintGCDetails
   -Dkey=value -Dnumbers="one two three"


#Stop all Services
$SPARK_HOME/sbin/stop-all.sh &&
   $HADOOP_COMMON_HOME/sbin/stop-yarn.sh &&
   $HADOOP_COMMON_HOME/sbin/stop-dfs.sh &&
   $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh --config
   $HADOOP_CONF_DIR stop historyserver &&
   $SPARK_HOME/sbin/stop-history-server.sh

#Start all Services
$HADOOP_COMMON_HOME/sbin/start-dfs.sh &&
   $HADOOP_COMMON_HOME/sbin/start-yarn.sh &&
   $HADOOP_HOME/sbin/mr-jobhistory-daemon.sh --config
   $HADOOP_CONF_DIR start historyserver &&
   $SPARK_HOME/sbin/start-all.sh &&
   $SPARK_HOME/sbin/start-history-server.sh
```

1.1 A Few Hints for Installation for OpenStack Installation

1. Install these sdks by cloning the respective git repositores and creating a fork on github for easy development and deployability back to the cluster for faster compilation and deployment

2. Externalise the configuration folder for spark and hadoop for convenience so that configuration is not erased when you deploy new builds.

3. Create a snapshot instance on openstack with this installed VM image after making the configurations. Copy to ssh key on to your remote machine to local machine.

4. After making these changes deploy a new instances and create entries for them in the slaves configuration file and establish that there is serverless ssh access between the nodes as hadoop and spark scripts rely on it to excute commands across the cluster.

2 Source Code

1. Class for Cluster, ClusteredPoint, KMeansClusterSummary A Cluster represents an indvidual cluster and set of ClusteredPoints associated with it.

   The KMeansClusterSummary has methods takes in the list of centers and data points and has methods to compute the WSSE and the DaviesBouldenIndex score for the resulting clusters. This class is to build clusters locally based on specified sample percentage extracted from the RDDs to run the cluster validation index locally.

```
package ie.ncirl.thesis.clustering.modelselection

import ie.ncirl.thesis.clustering.evaluation.DaviesBouldinIndex
import org.apache.spark.mllib.clustering.{KMeans,
   VectorWithNorm}
import org.apache.spark.mllib.linalg.{Vector => MlLibVector}
import org.apache.spark.rdd.RDD

case class Cluster(val index:Int, val centroid:VectorWithNorm,
   val points:Vector[ClusteredPoint]) {

  val size = points.size

  val totalDistanceToCentroid =
     computeIntraClusterDistanceToCentroid(points)

  val averageDistanceToCentroid = totalDistanceToCentroid/size

  def
     computeIntraClusterDistanceToCentroid(points:Vector[ClusteredPoint]):
     Double ={
    points.map(cp=> cp.distanceToCentroid).foldLeft(0.0)(_ + _)
  }
}

/**
  *
  * @param clusterIndex the index of the cluster to which this
     point has been assigned to
  * @param features the features vector for the point
  */
case class ClusteredPoint(clusterIndex: Int,features:
   MlLibVector,
                     distanceToCentroid: Double, norm: Double) {
  override def toString: String = {
    s"($clusterIndex,$features, $distanceToCentroid)"
  }

}

class KMeansClusterSummary(val clusterCentres:
```

```
  Array[VectorWithNorm],val data:Vector[VectorWithNorm] ) {

/**
  *
  * @param clusterCentres
  * @param data : the RDD from which the sample will be taken
  * @param sampleSize the size of the sample to be taken from
     the
  */
def this(clusterCentres : Array[VectorWithNorm],
   data:RDD[MlLibVector], sampleSize:Int){
  this(clusterCentres,data.takeSample(false, sampleSize).map(p
     => new VectorWithNorm(p)).toVector)
}

val clusterSize:Int = clusterCentres.size
val clusters : List[Cluster] = computeClusters()

val dbIndex : Double = getDaviesBouldinIndex()

/*
compute clusters from the given unlabelled points
*/
private def computeClusters(): List[Cluster] = {
  data.map(p => predictCluster(p))
    .groupBy(cp => cp.clusterIndex)
    .transform((clusterIndex, points)=> new
       Cluster(clusterIndex,clusterCentres(clusterIndex),
       points))
    .values.toList
}

/*
  the nearest cluster center is predicted based on the
     euclidean distance (l2-norm)
  and the ClusteredPoint is assigned to that cluster
 */
private def
   predictCluster(point:VectorWithNorm):ClusteredPoint = {
  val (clusterIndex, distanceToCentre) =
     KMeans.findClosest(clusterCentres, point)
  new ClusteredPoint(clusterIndex,point.vector,
     distanceToCentre, point.norm)
}


private def getDaviesBouldinIndex(): Double = {
  new DaviesBouldinIndex(clusters).computeIndex()
}
```

```
}
```

2. Class for DaviesBouldinIndex (adapted the Java implementation in the Java based Machine Learning JSAT package to a Scala Implemenation for Spark [1]

```
package ie.ncirl.thesis.clustering.evaluation

import ie.ncirl.thesis.clustering.modelselection.Cluster
import org.apache.spark.mllib.util.MLUtils

class DaviesBouldinIndex(clusters:List[Cluster]){

 def computeIndex(): Double ={

  var dbIndex = 0.0

  for (i <- 0 to clusters.size-1){

   var maximumPenalty = Double.NegativeInfinity;

   for (j <- 0 to clusters.size-1){

    if(i != j) {

     val ithCluster = clusters(i)
     val jthCluster = clusters(j)

     val distanceBetweenClusterCentroids =
      MLUtils.fastSquaredDistance(ithCluster.centroid.vector,
         ithCluster.centroid.norm,
        jthCluster.centroid.vector, jthCluster.centroid.norm)

     val penalty = (clusters(i).averageDistanceToCentroid
      + clusters(j).averageDistanceToCentroid ) /
         distanceBetweenClusterCentroids

     maximumPenalty = Math.max(maximumPenalty, penalty);
    }
   }

   dbIndex += maximumPenalty;

  }

  return dbIndex/clusters.size
 }

}
```

[1] https://github.com/EdwardRaff/JSAT

3. Clusterer Evaluator : This class gathers the list of clusters

```
package ie.ncirl.thesis.clustering.modelselection

import ie.ncirl.thesis.clustering.evaluation.ClusterScore

case class ClusteringResultsAggregator(val assumedClusterSizes:
   Seq[Int], val samplesTaken: Int,
                              val samplingPercentages:
                                 Seq[Double], val
                                 clusteringResults:
                                 Array[ClusteringResult]) {

 def computeClusterScores(): Array[ClusterScore] = {


   var clusterScores = new
      Array[ClusterScore](assumedClusterSizes.size-1)

   for (numberOfAssumedClusters <- assumedClusterSizes) {

     for (samplePercentage <- samplingPercentages) {

       //filter all results by cluster and sampling percentage
       val resultsForClusterSizeAndSamplePercentage =
        clusteringResults.filter(result => (result.clusterSize
           == numberOfAssumedClusters) &&
                                 (result.samplePercentage ==
                                    samplePercentage) )

       //make sure we have thirty samples for each cluster size
          and sample percentage
       assert(resultsForClusterSizeAndSamplePercentage.size ==
          samplesTaken)

       //transform each clustering result to collection of
          dbIndexScore and aggregate and divide by sample count
       val meanDBIndexScore =
          (resultsForClusterSizeAndSamplePercentage.map(_.dbIndex).foldLeft(
          + _)) / samplesTaken

       //since wsse score is computed on all data its same for
          across all samples so we just take the first
       val clusterScore = ClusterScore(numberOfAssumedClusters,
          samplePercentage, meanDBIndexScore,
          resultsForClusterSizeAndSamplePercentage(0).wsse)

       clusterScores = clusterScores :+ clusterScore
     }
   }
```

```
    clusterScores
  }

}
```

The above classes were initially implemented in the Spark framework within the org.apache.spark.mllib.clustering package. To improve the development cycle time they have been implemented outside the framework, as compiling and creating a spark build is very time taking.This necessiated changing the access modifiers a few convenience methods on KMeans and MLUtils. These package names can be refactored to be added to be integrated again into the framework.

4. Clusterer for Iris Data Set - used for evaluating the implemented methodology in spark local mode

```
package ie.ncirl.thesis

import
   ie.ncirl.thesis.clustering.modelselection.{ClusterValidator,
   ClusteringResultsAggregator}
import org.apache.log4j.Logger
import org.apache.spark.mllib.linalg._

import scala.io.Source

object IrisKMeansClusterer {
  def main(args: Array[String]) {

    SparkLogger.setLogLevels()
    val logger: Logger = Logger.getLogger(this.getClass)

    import org.apache.spark.sql.SparkSession

    val spark = SparkSession
      .builder()
      .appName("IrisLocal")
      .master("local[2]")
      .getOrCreate()

    //load iris from the resources folder
    val lines =
       Source.fromInputStream(this.getClass().getClassLoader().getResourceAs
    val data = spark.sparkContext.parallelize(lines.toList)

    val parsedData = data.map(s =>
       Vectors.dense(s.split(',').map(_.toDouble))).cache()

    // Cluster the data into two classes using KMeans

    val numIterations = 20
```

```
   //explicity define doubles as using an implicit range
      creates floats with large precision
   val samplePercentages = Seq(0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
      0.7, 0.8, 0.9 , 1.0)
   val numberOfSamplesTaken = 30
   val assumedClusterSizes = Seq(2,3,4,5,6,7,8,9,10)


   val cv = new
      ClusterValidator(assumedClusterSizes,numberOfSamplesTaken,samplePerce

   val clusteringResults = cv.computeClusterResults()
   logger.debug(s"The size of sample Results is
      ${cv.computeClusterResults().size}")
   logger.info(s"ClusterCount, SamplePercentage , SampleNumber
      , SampleCount , ClusterSize , WSSE , DbIndex")

   val resultsAggregator = new
      ClusteringResultsAggregator(assumedClusterSizes,
      numberOfSamplesTaken,samplePercentages,clusteringResults.toArray)
   resultsAggregator.computeClusterScores().filter(_
      !=null).foreach(println)

  }
}
```

5. Clusterer for GoWalla Data Set - used for evaluating the implemented methodology in spark local mode

```
package ie.ncirl.thesis

import
   ie.ncirl.thesis.clustering.modelselection.{ClusterValidator,
   ClusteringResultsAggregator}
import org.apache.log4j.Logger
import org.apache.spark.mllib.linalg.Vectors


/**
  * Created by vvs on 18/08/16.
  */
object GoWallaDataSetClusterer extends App {
  SparkLogger.setLogLevels()
  val logger: Logger = Logger.getLogger(this.getClass)

  import org.apache.spark.sql.SparkSession

  val spark = SparkSession
    .builder()
    .appName("GoWalla")
```

```
  .getOrCreate()

//load gowalla data
//[user] [check-in time]  [latitude] [longitude] [location id]
val gowallaData= spark.sparkContext
  .textFile("data/gowalla.data")
  .map{ line =>
     val buffer = line.split('\t').toBuffer
     //remove all attributes except lat long
     buffer.remove(0,2)
     buffer.remove(buffer.length-1)
     val vector = Vectors.dense(buffer.map(_.toDouble).toArray)
     (vector)
  }.cache()

gowallaData.take(1000).foreach(println)
println(gowallaData.count)

val numIterations = 20


//explicity define doubles as using an implicit range creates
   floats with large precision
val samplePercentages = Seq(0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
   0.7, 0.8, 0.9 , 1.0)
// A sample is a specified percentage of the data and is
   taken over 30 times.
val numberOfSamplesTaken = 30
val assumedClusterSizes = Seq(2,3,4,5,6,7,8,9,10)


val cv = new
   ClusterValidator(assumedClusterSizes,numberOfSamplesTaken,samplePercent

val clusteringResults = cv.computeClusterResults()
logger.debug(s"The size of sample Results is
   ${cv.computeClusterResults().size}")
logger.info(s"ClusterCount, SamplePercentage , SampleNumber ,
   SampleCount , ClusterSize , WSSE , DbIndex")

val resultsAggregator = new
   ClusteringResultsAggregator(assumedClusterSizes,
   numberOfSamplesTaken,samplePercentages,clusteringResults.toArray)
resultsAggregator.computeClusterScores().filter(_
   !=null).foreach(println)


spark.stop()
}
```

6. Clusterer for KDD Data Set - used for evaluating the implemented methodology in

spark local mode

```
package ie.ncirl.thesis

import
   ie.ncirl.thesis.clustering.modelselection.ClusteringResult
import org.apache.log4j.Logger
import org.apache.spark.mllib.clustering.{KMeans,
   KMeansClusterSummary, VectorWithNorm}
import org.apache.spark.mllib.feature.StandardScaler
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.stat.Statistics
import org.apache.spark.sql.SQLContext
import org.apache.spark.{SparkConf, SparkContext}

object KDD1999DataClusterer extends App{
  SparkLogger.setLogLevels()
  val logger:Logger = Logger.getLogger(this.getClass)

  val sc = new SparkContext(new
     SparkConf().setAppName("KMeans"))
  val sqlContext = new SQLContext(sc)

  //Load and parse the data
  var dataUrl = "hdfs:///user/hduser/kddcup.data"
  var numberOfClusters= 2

  if(args.size==2) {
    if (args(0).equalsIgnoreCase("sample")) {
      dataUrl = "hdfs:///user/hduser/kddcup.data_10_percent"
    }
    else if (args(0).equalsIgnoreCase("full")) {
      dataUrl = "hdfs:///user/hduser/kddcup.data"
    }

    numberOfClusters= Integer.parseInt(args(1))
  }else{
    println("Default Arguments Not Met Pass")
    sc.stop()
  }

  val rawData = sc.textFile(dataUrl)
  val labelsAndData = rawData.map { line =>
    val buffer = line.split(',').toBuffer
    buffer.remove(1, 3)
    val label = buffer.remove(buffer.length - 1)
    val vector = Vectors.dense(buffer.map(_.toDouble).toArray)
    (label, vector)
  }

  val data = labelsAndData.values.cache()
```

```
  //Summary statistics before scaling
  val stats = Statistics.colStats(data)
  logger.debug("Statistics before scaling")
  logger.debug(s"Max : ${stats.max}, Min : ${stats.min}, and
     Mean : ${stats.mean} and Variance : ${stats.variance}")

  val scaler = new StandardScaler(withMean = true, withStd =
     true).fit(data)
  val scaledData = scaler.transform(data).cache()
  val scaledStats = Statistics.colStats(scaledData)
  logger.debug("Statistics after scaling")
  logger.debug(s"Max : ${scaledStats.max}, Min :
     ${scaledStats.min}, and Mean : ${scaledStats.mean} " +
    s"and Variance : ${scaledStats.variance}")

  var clusteringResults = Vector[ClusteringResult]()

  for( sampleSize <- 10 to 20 by 10){

    val sampleData = scaledData.takeSample(false,
       (scaledData.count()/sampleSize).toInt).toVector.map(p =>
       new VectorWithNorm(p))

    val kmeans = new KMeans()
      .setK(numberOfClusters)
      .setMaxIterations(50)
      .setInitializationMode(KMeans.K_MEANS_PARALLEL) //KMeans||

    val model = kmeans.run(scaledData)

    val clusterCentersWithNorms = model.clusterCenters.map(p =>
       new VectorWithNorm(p))
    val clusterSummary: KMeansClusterSummary = new
       KMeansClusterSummary(clusterCentersWithNorms, sampleData)
    val wsse = model.computeCost(scaledData)

    clusteringResults = clusteringResults :+
       ClusteringResult(sampleSize,numberOfClusters,wsse,
       clusterSummary.dbIndex)

  }

  for(clusteringResult <- clusteringResults){
    logger.info(clusteringResult)
  }


}
```

7. Build Sbt for the application project using the Spark Custom Build, making changes

to spark parent pom.xml to also reference the local maven repo for development builds.

```
name := "UnsupervisedLearning"

version := "1.0"

scalaVersion := "2.11.8"

resolvers += Resolver.mavenLocal

lazy val root = (project in file("."))
  .settings(
    name := "my-project-name",
    version := "0.1",
    scalaVersion := "2.11.8",
    libraryDependencies ++= Seq(
      //"org.apache.spark" %% "spark-core" % "1.6.1" %
         "provided",
      //"org.apache.spark" %% "spark-mllib" % "1.6.1" %
         "provided",
      "org.apache.spark" %% "spark-core" % "2.1.0-SNAPSHOT",
      "org.apache.spark" %% "spark-mllib" % "2.1.0-SNAPSHOT",
      "org.slf4j" % "slf4j-simple" % "1.7.14",
      "io.continuum.bokeh" %% "bokeh" % "0.6"
/*
      "org.apache.spark" %% "spark-core" % "2.0.0",
      "org.apache.spark" %% "spark-mllib" % "2.0.0",
*/

    ),
    SparkDeployerPlugin.localModeSettings
  )

libraryDependencies += "net.sourceforge.f2j" %
   "arpack_combined_all" % "0.1"
libraryDependencies += "com.github.fommil.netlib" %
   "netlib-native_ref-linux-x86_64" % "1.1" classifier "natives"
libraryDependencies += "com.github.fommil.netlib" %
   "netlib-native_system-linux-x86_64" % "1.1" classifier
   "natives"
```

8. Dynamic Model Evaluator

```
// scalastyle:off

package org.apache.spark.ml.evaluation

import org.apache.spark.internal.Logging
import org.apache.spark.mllib.clustering.KMeansModel
import org.apache.spark.mllib.clustering.VectorWithNorm
```

```
import org.apache.spark.rdd.RDD
import scala.collection.immutable.Vector
class DynamicModelEvaluator(maxIterations: Int) extends Logging
   with Serializable{

 var triggerEnabled = false

 var incrementalModels:Vector[KMeansModel] =
    Vector[KMeansModel]()

 var dynamicModelEvaluationTriggered = false

 val modelEvaluationFrequency:Int = 10

 val minIterationsToTriggerDynamicModelEvaluation:Int = 100

 val modelIterationTriggerPercentage = .3

 def evaluateThisIteration(iteration: Int): Boolean = {
  if(!triggerEnabled){
    triggerEnabled = checkTrigger(iteration)
    return triggerEnabled
  }else {
    //once trigger is enabled we just have to check
    return iteration % modelEvaluationFrequency == 0
  }


 }

 /**
  * overloaded for sampling based evaluation
  *
  * @param incrementalModels
  * @param data
  * @return
  */
 def evaluate(data:RDD[VectorWithNorm], metric:String =
    "cost"):Boolean = {
  val modelCount:Int = incrementalModels.size
  logInfo(s"Incremental model count is $modelCount")

  val costs = new Array[Double](incrementalModels.size)
  var costIndex = 0

  for((currentModel,index) <- incrementalModels.zipWithIndex){

    //we only want to compute the index for the last 3 models
    //therefore if 4 models stored, we only compute for index
    if((index > modelCount - 3)){
```

```
      costs(costIndex) =
         currentModel.computeCost(data.map(point=>point.vector))
      costIndex+=1
    }
  }

  logInfo(s"Costs are costs of model 1 : ${costs(0)},
     ${costs(1)}, ${costs(2)}")
  //return true if costs are decreasing
  return costs(0) < costs (1) && costs(1) < costs(2)
 }

 def checkTrigger(iteration:Int):Boolean = {
  if (maxIterations >=
     minIterationsToTriggerDynamicModelEvaluation
   && iteration >= ( maxIterations *
      modelIterationTriggerPercentage)) {
   return true
  }

  return false
 }

 def addModel(kMeansModel: KMeansModel): Unit ={
  incrementalModels :+ kMeansModel
 }

 def evaluateIncrementalModels(): Boolean = {
  return incrementalModels.size >=3
 }

}

// scalastyle:on
```

This class is an abstraction based on the concepts modeled in the dynamic model evaluation framework implemented in the "Learning to Learn with Spark" theis TN (2015). The original implementation for that paper by extending the StochasticGradientDescent optimisation part of SVM algorithm, via an additional paramater for passing in the test dataset for evaluating the learning performance during 10

The spark project has moved at a fast pace since and is now at version 2.0 and there have been significant changes and improvements in the implementation. Spark now provides the concepts of ML pipeline modeled after the sci-kit learn python package and provides uniform abstractions like Transformers, Estimators, Evaluator, Pipeline to provide a unified API to build machine learning pipelines on the spark framework.

Spark MLlib package itself has been renamed from mllib to ml and the core abstractions have now moved from RDD to Dataframes to Datasets.

The Dynamic Model Evaluator was initially proposed to modeled on the Evaluator abstraction inbuilt in spark, however the contract of the abstraction werent matching the requirements of the project and hence inbuilt Evaluator couldnt be extended.

9. KMeans (extended to included Dynamic Model Evaluator) The changes made to KMeans implmentation in spark mllib to integrate this dynamic evaluator However as the project progressed, due to the lack of inbuilt measures for cluster evaluation within spark project other than the Weighted Sum of Squared Errors which was monotonic in nature, the project focus shifted to implementing a sampling based internal clustering validation indexand related classes for building a sample based cluster validator as an additional metric to evaluate a clustering algoirthm. The source file of KMeans.scala with the portions with the relevant modifications integrating the dynamic model evaluator have been included because of its length of the orginal source extending 18 pages from the original spark implementation. The relevant portion contribute to the code has been marked in between "#####". The full source is avialable at these links [2] and [3].

```
class KMeans private(
                private var k: Int,
                private var maxIterations: Int,
                private var runs: Int,
                private var initializationMode: String,
                private var initializationSteps: Int,
                private var epsilon: Double,
                private var seed: Long) extends Serializable
                   with Logging {

 //###############################################
  val dynamicModelEvaluator: DynamicModelEvaluator = new
     DynamicModelEvaluator(maxIterations)
 //###############################################
  .........


 /**
   * Implementation of K-Means algorithm.
   */
 private def runAlgorithm(
                    data: RDD[VectorWithNorm],
                    instr:
                       Option[Instrumentation[NewKMeans]]):
                       KMeansModel = {

  val observationsCount = data.count()

  //###############################################
  //  take a testData sample at the start for dynamic model
     evaluation before the iterations begin
```

[2] https://github.com/vidyasankarv/spark/blob/thesis/mllib/src/main/scala/org/apache/spark/mllib/clustering/KMean
[3] https://github.com/vidyasankarv/UnsupervisedLearning

```
//   so that we evaluate on same set of points across the
   iterations
//   Will changing to retrieve different points at different
   times help ?
//
val testData = data.takeSample(false, (observationsCount *
   .10).toInt, 123456)
//##########################################


val sc = data.sparkContext

val initStartTime = System.nanoTime()

// Only one run is allowed when initialModel is given
val numRuns = if (initialModel.nonEmpty) {
  if (runs > 1) logWarning("Ignoring runs; one run is
     allowed when initialModel is given.")
  1
} else {
  runs
}

val centers = initialModel match {
  case Some(kMeansCenters) =>
    Array(kMeansCenters.clusterCenters.map(s => new
       VectorWithNorm(s)))
  case None =>
    if (initializationMode == KMeans.RANDOM) {
      initRandom(data)
    } else {
      initKMeansParallel(data)
    }
}
val initTimeInSeconds = (System.nanoTime() - initStartTime)
   / 1e9
logInfo(s"Initialization with $initializationMode took " +
   "%.3f".format(initTimeInSeconds) +
  " seconds.")

val active = Array.fill(numRuns)(true)
val costs = Array.fill(numRuns)(0.0)

var activeRuns = new ArrayBuffer[Int] ++ (0 until numRuns)
var iteration = 0

val iterationStartTime = System.nanoTime()

instr.foreach(_.logNumFeatures(centers(0)(0).vector.size))

// Execute iterations of Lloyd's algorithm until all runs
```

```
    have converged
  while (iteration < maxIterations && !activeRuns.isEmpty) {
   type WeightedPoint = (Vector, Long)
   def mergeContribs(x: WeightedPoint, y: WeightedPoint):
      WeightedPoint = {

    //AXPY constant times a vector plus a vector.
    //* y += a * x def axpy(a: Double, x: Vector, y: Vector):
       Unit = { .. */
    axpy(1.0, x._1, y._1)
    //return the combined vector and the sum of their contrib
       ?
    (y._1, x._2 + y._2)
   }

   //for runs = 1, k =2, activeRuns is an
      Array[Array[VectorWithNorm]] and is of size 1,
   // and that activeCenter(0).size has 2 points representing
      the 2 cluster centers of the first Array
   val activeCenters = activeRuns.map(r => centers(r)).toArray

   //A cost Accumulator for each run
   val costAccums = activeRuns.map(_ => sc.doubleAccumulator)

   //broadcast the active centers to each partitions
   val bcActiveCenters = sc.broadcast(activeCenters)

   // Find the sum and count of points mapping to each center
   //example of map partition function along with the its use
      of broadcast variables and accumulators.
   val totalContribs = data.mapPartitions { points =>
    val thisActiveCenters = bcActiveCenters.value
    val runs = thisActiveCenters.length
    val k = thisActiveCenters(0).length
    val dims = thisActiveCenters(0)(0).vector.size

    //A nested array, one for each run, to hold the sum of
       all points belonging to the cluster in each run
    val sums = Array.fill(runs, k)(Vectors.zeros(dims))

    val counts = Array.fill(runs, k)(0L)


    //put each point through various "runs"
    points.foreach { point =>

     //so this is where we make use of the runs on each
        partition
     (0 until runs).foreach { i =>

      //for each run's active centers, find the closest
```

```
          center to that point
        val (bestCenter, cost) =
          KMeans.findClosest(thisActiveCenters(i), point)

        //add cost to cost accumulator for that run
        costAccums(i).add(cost)

        //accumulating the values of all the points in the
          cluster
        //to find the new cluster center by averaging them
        //sums(i) - sums vector for that run and get that sum
          vector for the nearest cluster for this point
        //sum holds a reference to say sums(run0) and its best
          center - the sum of all vectors in that cluster
        val sum = sums(i)(bestCenter)
        //add the sum of this point vector to that sum
        axpy(1.0, point.vector, sum)

        //increase the count of best centers run
        counts(i)(bestCenter) += 1
      }
    }

    //yield the tuple for each run
    val contribs = for (i <- 0 until runs; j <- 0 until k)
      yield {
      ((i, j), (sums(i)(j), counts(i)(j)))
    }

    contribs.iterator
  }.reduceByKey(mergeContribs).collectAsMap()

  bcActiveCenters.destroy(blocking = false)

  // Update the cluster centers and costs for each active run
  // this is the update step and checking in the center has
    changed
  for ((run, i) <- activeRuns.zipWithIndex) {
    var changed = false
    var j = 0
    while (j < k) {
      val (sum, count) = totalContribs((i, j))
      if (count != 0) {
        scal(1.0 / count, sum)
        val newCenter = new VectorWithNorm(sum)
        if (KMeans.fastSquaredDistance(newCenter,
          centers(run)(j)) > epsilon * epsilon) {
          changed = true
        }
        centers(run)(j) = newCenter
      }
```

```
      j += 1
    }
    if (!changed) {
      active(run) = false
      logInfo("Run " + run + " finished in " + (iteration +
         1) + " iterations")
    }
    costs(run) = costAccums(i).value
  }

  activeRuns = activeRuns.filter(active(_))
  iteration += 1

  //######################
  /** code for dynamic model evaluation added */
  if
     (dynamicModelEvaluator.evaluateThisIteration(iteration))
     {
    logInfo(s"Evaluating Iteration $iteration")
    //since its a fixed size fifo Queue,
    //when the latest increment of the model is pushed the
       oldest one gets pushed out of the queue
    dynamicModelEvaluator.addModel(new
       KMeansModel(centers(0).map(_.vector)))

    //i.e if there are 3 models, which there will always be
       after the fiftieth iteration
    if (dynamicModelEvaluator.evaluateIncrementalModels()) {
      dynamicModelEvaluator.evaluate(data)
    }

  }
  //######################

}

val iterationTimeInSeconds = (System.nanoTime() -
   iterationStartTime) / 1e9
logInfo(s"Iterations took " +
   "%.3f".format(iterationTimeInSeconds) + " seconds.")

if (iteration == maxIterations) {
  logInfo(s"KMeans reached the max number of iterations:
     $maxIterations.")
} else {
  logInfo(s"KMeans converged in $iteration iterations.")
}

val (minCost, bestRun) = costs.zipWithIndex.min

logInfo(s"The cost for the best run is $minCost.")
```

```
    new KMeansModel(centers(bestRun).map(_.vector))
  }

}
```

3 Code References

1. Ryza et al. (2015)
2. Nicolas (2015)
3. Karau et al. (2015)
4. White (2012)

References

Karau, H., Konwinski, A., Wendell, P. and Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*, " O'Reilly Media, Inc.".

Nicolas, P. R. (2015). *Scala for Machine Learning*, Packt Publishing Ltd.

Ryza, S., Laserson, U., Owen, S. and Wills, J. (2015). *Advanced Analytics with Spark: Patterns for Learning from Data at Scale*, " O'Reilly Media, Inc.".

TN, S. B. (2015). *Learning to learn with spark*, Master's thesis, National College of Ireland.

White, T. (2012). *Hadoop: The definitive guide*, " O'Reilly Media, Inc.".