Introduction to Spark Machine Learning

October 9, 2017

1 Exercises

1. check out the source code.

```
$ cd examples/spark-ml-examples
```

1.1 Tweet classifier

The school have collected some twitter data on the keyword "jae". However the term "jae" could refer to some Korean celebrities having "jae" as part of their names, or to joint admission exercise in Singapore. The task here is to build a classifier to differentiate the KPOP tweets mentioning "jae" as a person's name or otherwise.

For example, the following tweet message falls into the category of Korean Pop because it seems talking about a person's name

```
crazy cool   jae s lee's pic of street singer reflected in raindrops
 tuesday on 2nd ave
```

On the other hand, the following tweet is not revelant to KPOP.

```
accident closes jae valley rd drivers advised to avoid area seek
alternate routes
```

To achieve the goal, we need to develop a classifier, which is a supervised machine learning technique. In this example, we consider using Support Vector Machine (SVM) as the classifier algorithm. On the higher level, we need to "train" the model with some manually labelled data and perform some tests against the trained model. As part of the input requirement the SVM expect the input data to represented as a label (either yes or no, 1 or 0) accompanied by the feature vector. The feature vector is a vector of values which uniquely differentiate one entry from another ideally. In the machine learning context, features have to be fixed by the programmers.

For the ease of illustration, we use a vector of hash codes derived from the words appearing in the tweet. Consider the following source code in `src/main/scala/TweetSVMFilter.scala`

```
object TweetSVMFilter {

  val vector_fixed_size = 30
  // fixed the size of each vector.
  // if vectors have different sizes, the gradient descent algorithm will fail
  // cut off if it exceeds, pad zeros if it has less than 30 elements

  def hash(str:String):Int = str.toList.foldLeft(2147483647)((h,c) => 31*h + c)

  def to_words(tweet:String):List[String] = tweet.split(" ").toList

  def pad_cap(xs:List[Double],size:Int):List[Double] = xs match
  {
    case Nil if size > 0 => List.fill(size)(0.0) // fill the rest with 0.0
    case Nil             => Nil
    case (y::ys) if size == 0 => Nil
    case (y::ys)              => y::(pad_cap(ys,size-1))
  }

  def main(args: Array[String]) {

    val sc = new SparkContext("local", "shell")
    // Load training data in LIBSVM format.
    val posTXT:RDD[String] = sc.textFile("data/tweet/label_data/Kpop/*.txt") // .sample(fals
    val negTXT:RDD[String] = sc.textFile("data/tweet/label_data/othertweet/*.txt") // .sampl

    // convert the training data to labeled points
    val posLP:RDD[LabeledPoint] = posTXT.map( (twt:String) =>
    {
      val ws = to_words(twt).map(w => hash(w).toDouble)
      LabeledPoint(1.0, Vectors.dense(pad_cap(ws ,vector_fixed_size).toArray))
    })
    val negLP:RDD[LabeledPoint] = negTXT.map( (twt:String) =>
    {
      val ws = to_words(twt).map(w => hash(w).toDouble)
      LabeledPoint(0.0, Vectors.dense(pad_cap(ws ,vector_fixed_size).toArray))
    })
    val data = negLP ++ posLP

    // Split data into training (60%) and test (40%).
    val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
    val training = splits(0).cache()
    val test = splits(1)

    // Run training algorithm to build the model
    val numIterations = 10
    val model = SVMWithSGD.train(training, numIterations)

    // Clear the default threshold.
    model.clearThreshold()

    // Compute raw scores on the test set.
    val scoreAndLabels = test.map { point =>
      val score = model.predict(point.features)
      (score, point.label)
    }

    // Get evaluation metrics.
    val metrics = new BinaryClassificationMetrics(scoreAndLabels)
    val auROC = metrics.areaUnderROC()

    println("Area under ROC = " + auROC)
    sc.stop
  }
}
```

The `hash` function hashes a string value into an integer. The `to_words` function turns a sentence into words. The `pad_cap` function takes a list of double values as input `xs` and a size `size`. If the size of the input list exceeds the given size, it truncates the input list down to the size. If the size of the input falls below `size`, it pads the remaining spaces with zeros. This is to ensure all vectors to the SVM have the same size. In the `main` function, we first load the labeled data from the HDFS. We construct the labeled points from the input data. Note that the vector in a labeled point is constructed from the input text by taking the first 30 words from the text and converting each word into a hash code. In the following steps, we combine the positive and the negative labeled points. By randomly split the labeled points into two sets, namely 60% for the model construction and 40% for the testing.

1. To compile

   ```
   $ sbt package
   ```

2. To execute

   ```
   $ spark-submit --class TweetSVMFilter\
   target/scala-2.11/spark-ml-examples_2.11-0.1.jar
   ```

   We will observe the output

   ```
   Area under ROC = 0.5517569981668705
   ```

1.2 Exercises

The accuracy of the above SVM model is not very high. Using invidiual word as feature might not be a good idea.

- Instead of using single word, try to use 2-gram and 3-gram phases as features.
- Instead of taking all the 2-gram (or 3-gram) phases, we can run TF-IDF to select input key phrases as features.