Распределенные алгоритмы. Spark MLlib. Наивный байесовский классификатор

Содержание:

Наивный байесовский классификатор

Полиномиальная модель

Задача классификации в данном случае заключается в отнесении некоторого документа d к одному из классов c_i из заранее заданного множества C такого, что

$$c_i \in C, |C| = K, 1 \leq i \leq K,$$

где K – количество классов.

При использовании байесовского классификаторы определяется вероятность принадлежности документа d каждому классу из C. В качестве предсказания выбирается класс с наибольшей вероятностью.

$$P(c_i|d) \propto P(c_i) \cdot P(d|c_i)$$

Документ d можно представить в виде вектора термов:

$$\mathrm{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix},$$

где x_j – количественное значение терма t_j в документе (количество вхождений, TF-IDF или др.); M – количество термов (признаков).

В наивной форме вводится допущение о независимости термов, поэтому можно представить $P(d|c_i)$ как

$$P(\mathrm{x}|c) = \prod_{j=1}^{M} P(t_j|c)^{x_j}$$

Тогда общее выражение примет вид:

$$P(c_i|d) \propto P(c_i) \cdot \prod_{j=1}^{M} P(t_j|c_i)^{x_j}$$

Обучение

Обозначим множество документов с указанными действительными классами как

$$D = \{(c_1, \mathrm{x}_1), \dots, (c_N, \mathrm{x}_N)\},$$

где c_n – обозначение действительного класса документа $d_n \in D$ (целевое значение); x_n – векторное представление документа d_n; N – количество элементов выборки.

Обучение будет заключаться в вычислении оценок вероятностей $P(c_i)$ и $P(t_j|c_i)$.

Оценка вероятности, что документ будет отнесен к классу c_i:

$$\hat{P}(c_i) = \frac{N_i}{N},$$

где N_i – количество документов класса c_i; N – общее количество документов.

Оценка вероятности того, что в документе класса c_i встретится терм t_j:

$$\hat{P}(t_j|c_i) = \frac{n_i(t_j)}{\sum_{j=1}^{M} n_i(t_j)},$$

где $n_i(t_j)$ – количество терма t_j в документах класса c_i.

Если использовать сглаживание, чтобы избежать случаев с нулевой вероятностью, то

$$\hat{P}(c_i; \lambda) = \frac{N_i + \lambda}{N + K\lambda}$$

и

$$\hat{P}(t_j|c_i; \lambda) = \frac{n_i(t_j) + \lambda}{\sum_{j=1}^{M} n_i(t_j) + M\lambda}$$

Предсказание

В общем виде для полиномиальной модели выражение для предсказания класса документа имеет вид:

$$\hat{c} = c_{MAP} = \underset{\substack{c_i \in C \\ 1 \le i \le K}}{\operatorname{argmax}} \hat{P}(c_i) \cdot \prod_{j=1}^{M} \hat{P}(t_j|c_i)^{x_j}.$$

Операция сложения более предпочтительна, чем умножение, когда имеем дело с малыми значениями. Поэтому запишем предыдущее выражение следующим образом:

$$\hat{c} = \underset{\substack{c_i \in C \\ 1 \le i \le K}}{\operatorname{argmax}} \left[\ln \hat{P}(c_i) + \sum_{j=1}^{M} x_j \ln \hat{P}(t_j|c_i) \right].$$

В свою очередь натуральный логарифм от $\hat{P}(c_i; \lambda)$ можно разложить на отдельные составляющие:

$$\pi_i = \ln \hat{P}(c_i; \lambda) = \ln(N_i + \lambda) - \ln(N + K\lambda).$$

Аналогичным образом поступим с $\hat{P}(t_j|c_i;\lambda)$:

$$\theta_{ij} = \ln \hat{P}(t_j|c_i;\lambda) = \ln(n_i(t_j) + \lambda) - \ln\left(\sum_{j=1}^{M} n_i(t_j) + M\lambda\right).$$

Тогда предсказание принадлежности некоторого документа d, представленного в виде вектора признаков x, можно записать следующим образом:

$$\hat{c} = \underset{\substack{c_i \in C \\ 1 \le i \le K}}{\operatorname{argmax}} \left[\pi_i + \sum_{j=1}^{M} x_j \theta_{ij}\right]$$

или

$$\hat{c} = \underset{\substack{c_i \in C \\ 1 \le i \le K}}{\operatorname{argmax}} [\pi_i + \mathrm{x}^T \theta_i].$$

Алгоритмы распределенного обучения и предсказания в Spark MLlib

Обучение

Распределенный характер обучения подразумевает, что обучающие данные D разбиты на части, каждая из которых обрабатывается отдельно. Так как обучение байесовского наивного классификатора для полиномиальной модели сводится к подсчету оценок вероятностей $\hat{P}(c_i)$ и $\hat{P}(t_j|c_i)$, но необходимо определить n_i и N_i, где

$$\mathrm{n}_i = \begin{pmatrix} n_i(t_1) \\ \vdots \\ n_i(t_M) \end{pmatrix}.$$

Пусть исходное обучающее множество разбито на P частей, тогда для подсчета n_i и N_i необходимо вычислить n_i^p и N_i^p в каждой части p, $1 \le p \le P$, и просуммировать значения отдельно по каждому классу c_i, $1 \le i \le K$:

$$\mathrm{n}_i = \sum_{p=1}^{P} \mathrm{n}_i^p,$$

$$N_i = \sum_{p=1}^{P} N_i^p,$$

где

$$\mathrm{n}_i^p = \begin{pmatrix} n_i^p(t_1) \\ \vdots \\ n_i^p(t_M) \end{pmatrix}$$

и

$$N_i^p = \sum_{d \in D^p} 1\,(c = c_i)$$

Графически это можно изобразить, как представлено на рисунке 1.

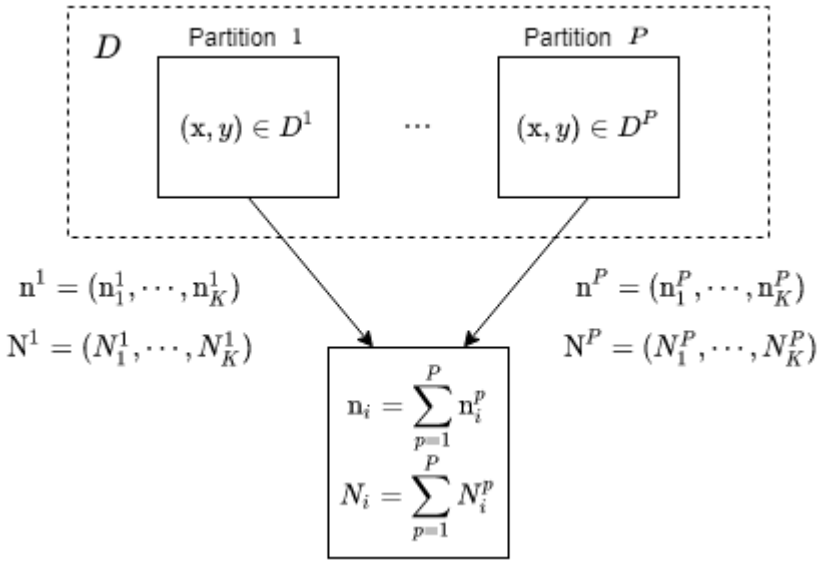

Рисунок 1. Распределенное вычисление n_i и N_i

Затем для каждого класса рассчитываем значения π_i и θ_i:

$$\pi_i = \ln \hat{P}(c_i) = \ln(N_i + \lambda) - \ln(N + K\lambda),$$

$$\theta_i = \begin{pmatrix} \theta_{i1} \\ \vdots \\ \theta_{iM} \end{pmatrix},$$

где

$$\theta_{ij} = \ln \hat{P}\left(t_j \middle| c_i\right) = \ln\left(n_i\left(t_j\right) + \lambda\right) - \ln\left(\sum_{j=1}^{M} n_i\left(t_j\right) + M\lambda\right)$$

и

$$N = \sum_{i=1}^{K} N_i$$

Конечный результат представляем в виде пары (π, Θ), где

$$\pi = \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_K \end{pmatrix} \text{ и } \Theta = \begin{pmatrix} \theta_{11} & \cdots & \theta_{1M} \\ \vdots & \ddots & \vdots \\ \theta_{K1} & \cdots & \theta_{KM} \end{pmatrix}.$$

Соответствующий алгоритм в Spark MLlib для DataFrame API приведен ниже.

Алгоритм 1. Наивный байесовский классификатор. Полиномиальная модель. Обучение

1	loadDistributed(D)	$D = \{(c_1, \mathrm{x}_1), \ldots, (c_N, \mathrm{x}_N)\}$
2	for p in $1..P$ in parallel:	
3	$\mathrm{n}^p \leftarrow \mathbf{0}^{M \times K}, N^p \leftarrow \mathbf{0}^K$	
4	for (i, x) in D^p:	i – класс документа
5	$\mathrm{n}_i^p \leftarrow \mathrm{n}_i^p + x$	$\mathrm{n}_i^p = \left(n_i^p(t_1), \ldots, n_i^p(t_M)\right)$
6	$N_i^p \leftarrow N_i^p + 1$	

7	collect$\left((\mathrm{n}^1, N^1), \ldots, (\mathrm{n}^P, N^P)\right)$	$\mathrm{n}^p = \left(\mathrm{n}_1^p, \ldots, \mathrm{n}_K^p\right), N^p = \left(N_1^p, \ldots, N_K^p\right)$
8	for p in $1..P$:	
9	$\mathrm{n}_i \leftarrow \mathrm{n}_i + n_i^p$	
10	$N_i \leftarrow N_i + N_i^p$	
11	$N \leftarrow N + N_i$ for i in $1..K$	
12	$\pi_{denom} = \ln(N + K\lambda)$	
13	initialize(π, Θ)	
14	for i in $1..K$:	
15	$\pi_i = \ln(N_i + \lambda) - \pi_{denom}$	
16	$\theta_{i,denom} \leftarrow \ln\left(\sum_{j=1}^{M} \mathrm{n}_{ij} + M\lambda\right)$	
17	for j in $1..M$:	
18	$\theta_{ij} \leftarrow \ln\left(\mathrm{n}_{ij} + \lambda\right) - \theta_{i,denom}$	
19	return(π, Θ)	

Предсказание

Для предсказания принадлежности документа d' к классу из C используется выражение

$$\hat{c} = \underset{\substack{c_i \in C \\ 1 \le i \le K}}{\operatorname{argmax}} \left[\pi_i + \mathrm{x'}^T \theta_i\right]$$

где x' – вектор термов документа d':

$$\mathrm{x}' = \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix}.$$

Алгоритм предсказания в Spark MLlib для DataFrame API приведен ниже.

Алгоритм 2. Наивный байесовский классификатор. Полиномиальная модель. Предсказание

1	loadDistributed$(D_{Features})$	$D_{Features} = \left\{\mathrm{x}_1, \ldots, \mathrm{x}_{N_{Features}}\right\}$
2	broadcast(π, Θ)	
3	for p in $1..P$ in parallel:	
4	$\mathrm{pred}^p \leftarrow \mathbf{null}^{\left\|D_{Features}^p\right\|}$	Инициализация массива предсказаний
5	for $(\mathrm{indx}, \mathrm{x})$ in $D_{Features}^p$:	indx – индекс
6	$\mathrm{prob} \leftarrow \mathrm{x}^T\Theta + \pi$	prob – вектор размерности K
7	$\mathrm{pred}_{\mathrm{indx}}^p \leftarrow \mathrm{argmax}(\mathrm{prob})$	
8	$\mathrm{pred} \leftarrow \mathrm{collect}(\mathrm{pred}^1, \ldots, \mathrm{pred}^P)$	
9	return pred	

Реализация алгоритмов в Spark MLlib

Листинг 1. Spark MLlib. Naïve Bayes. Multinomial Model. Обучение

```
@Since("1.5.0")
class NaiveBayes extends ProbabilisticClassifier {

  override protected def train(dataset: Dataset[_]): NaiveBayesModel = {
    trainWithLabelCheck(dataset, positiveLabel = true)
```

```
  }

  private[spark] def trainWithLabelCheck(
      dataset: Dataset[_],
      positiveLabel: Boolean): NaiveBayesModel = instrumented { instr =>

    val numFeatures = dataset.select(col($(featuresCol))).head().getAs[Vector](0).size
    val w = if (!isDefined(weightCol) || $(weightCol).isEmpty) lit(1.0) else col($(weightCol))

    // Aggregates term frequencies per label.
    // TODO: Calling aggregateByKey and collect creates two stages, we can implement something
    // TODO: similar to reduceByKeyLocally to save one stage.
    val aggregated = dataset.select(col($(labelCol)), w, col($(featuresCol))).rdd
      .map { row => (row.getDouble(0), (row.getDouble(1), row.getAs[Vector](2)))
      }.aggregateByKey[(Double, DenseVector, Long)]((0.0, Vectors.zeros(numFeatures).toDense,
0L))(
      seqOp = {
        case ((weightSum, featureSum, count), (weight, features)) =>
          requireValues(features)
          BLAS.axpy(weight, features, featureSum)
          (weightSum + weight, featureSum, count + 1)
      },
      combOp = {
        case ((weightSum1, featureSum1, count1), (weightSum2, featureSum2, count2)) =>
          BLAS.axpy(1.0, featureSum2, featureSum1)
          (weightSum1 + weightSum2, featureSum1, count1 + count2)
      }).collect().sortBy(_._1)

    val numSamples = aggregated.map( . 2. 3).sum
    val numLabels = aggregated.length
    val numDocuments = aggregated.map(_._2._1).sum

    val labelArray = new Array[Double](numLabels)
    val piArray = new Array[Double](numLabels)
    val thetaArray = new Array[Double](numLabels * numFeatures)

    val lambda = $(smoothing)
    val piLogDenom = math.log(numDocuments + numLabels * lambda)
    var i = 0
    aggregated.foreach { case (label, (n, sumTermFreqs, _)) =>
      labelArray(i) = label
      piArray(i) = math.log(n + lambda) - piLogDenom
      val thetaLogDenom = $(modelType) match {
        case Multinomial => math.log(sumTermFreqs.values.sum + numFeatures * lambda)
        case Bernoulli => math.log(n + 2.0 * lambda)
        case _ =>
          // This should never happen.
          throw new UnknownError(s"Invalid modelType: ${$(modelType)}.")
      }
      var j = 0
      while (j < numFeatures) {
        thetaArray(i * numFeatures + j) = math.log(sumTermFreqs(j) + lambda) - thetaLogDenom
        j += 1
      }
      i += 1
    }

    val pi = Vectors.dense(piArray)
    val theta = new DenseMatrix(numLabels, numFeatures, thetaArray, true)
    new NaiveBayesModel(uid, pi, theta).setOldLabels(labelArray)
  }
}
```

Листинг 2. Spark MLlib. Naïve Bayes. Multinomial Model. Предсказание

```
@DeveloperApi
abstract class ClassificationModel[FeaturesType, M <: ClassificationModel[FeaturesType, M]]
  extends PredictionModel[FeaturesType, M] with ClassifierParams {

  override def transform(dataset: Dataset[_]): DataFrame = {
    transformSchema(dataset.schema, logging = true)

    var outputData = dataset
```

```
    if (getPredictionCol != "") {
      val predUDF = udf { (features: Any) =>
          predict(features.asInstanceOf[FeaturesType])
        }
        predictUDF(col(getFeaturesCol))
      }
      outputData = outputData.withColumn(getPredictionCol, predUDF)
    }
    outputData.toDF
  }

  override def predict(features: FeaturesType): Double = {
    raw2prediction(predictRaw(features))
  }

  protected def predictRaw(features: FeaturesType): Vector

  protected def raw2prediction(rawPrediction: Vector): Double = rawPrediction.argmax
}


@Since("1.5.0")
class NaiveBayesModel private[ml] (
    @Since("1.5.0") override val uid: String,
    @Since("2.0.0") val pi: Vector,
    @Since("2.0.0") val theta: Matrix)
  extends ProbabilisticClassificationModel[Vector, NaiveBayesModel]
    with NaiveBayesParams with MLWritable {

  private def multinomialCalculation(features: Vector) = {
    val prob = theta.multiply(features)
    BLAS.axpy(1.0, pi, prob)
    prob
  }

  override protected def predictRaw(features: Vector): Vector = {
    $(modelType) match {
      case Multinomial =>
        multinomialCalculation(features)
      case Bernoulli =>
        bernoulliCalculation(features)
      case _ =>
        // This should never happen.
        throw new UnknownError(s"Invalid modelType: ${$(modelType)}.")
    }
  }
}
```

Список литературы

1. Text Classification using Naive Bayes by Hiroshi Shimodaira. URL: https://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn07-notes-nup.pdf
2. Chapter 13 Text classification and Naive Bayes // Introduction to Information Retrieval by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schutze, Cambridge University Press. 2008. URL: http://www-nlp.stanford.edu/IR-book/
3. Source Code of MLlib for DataFrame API. URL: https://github.com/apache/spark/tree/master/mllib/src/main/scala/org/apache/spark/ml