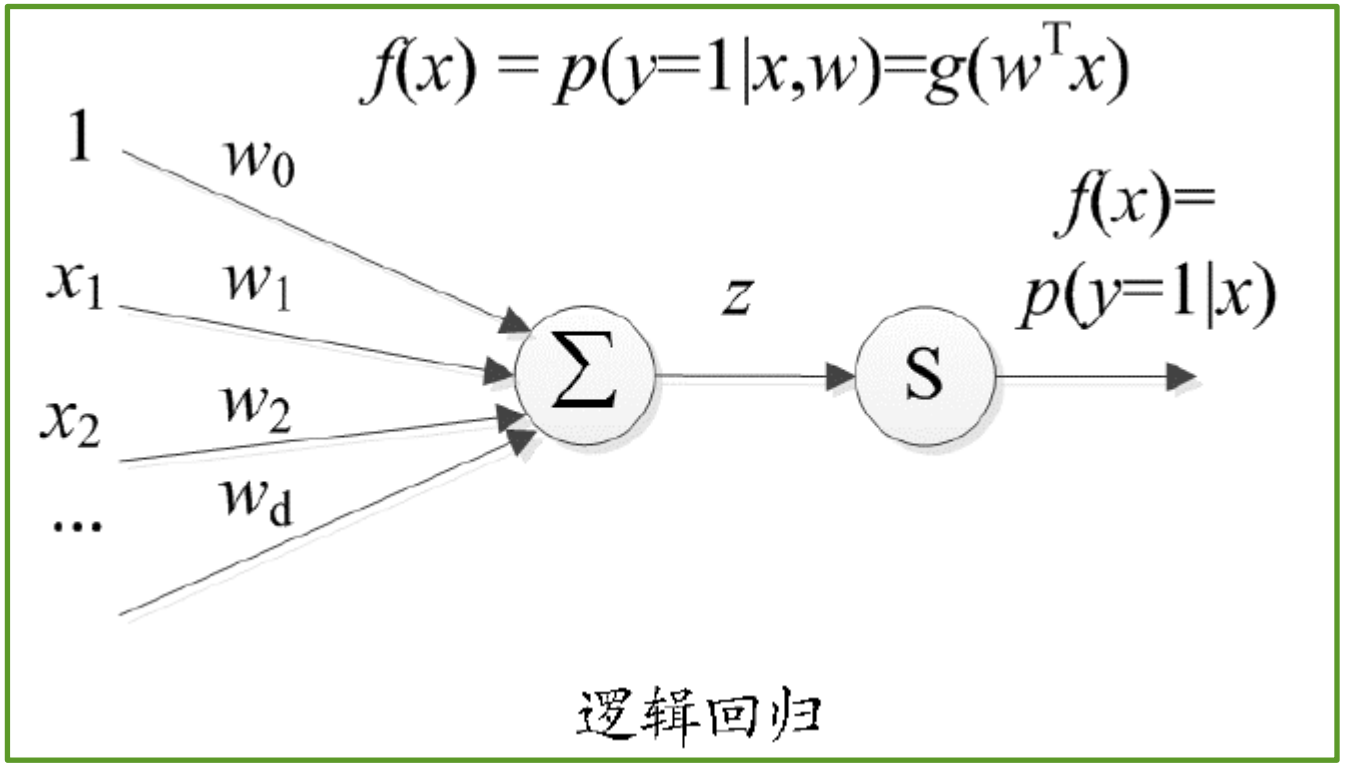
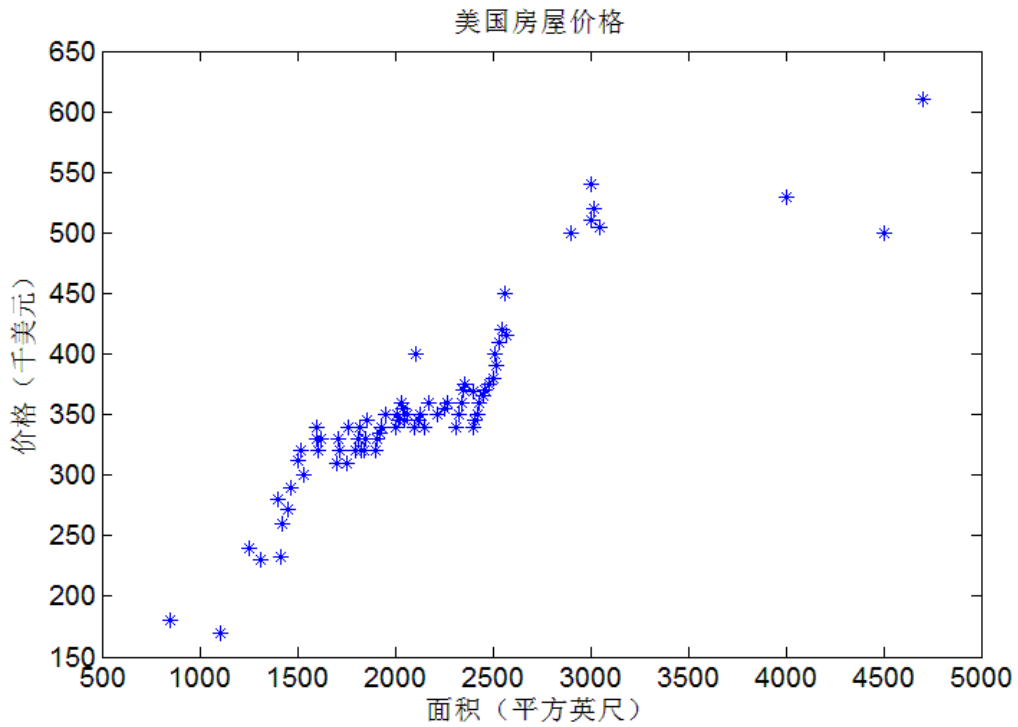


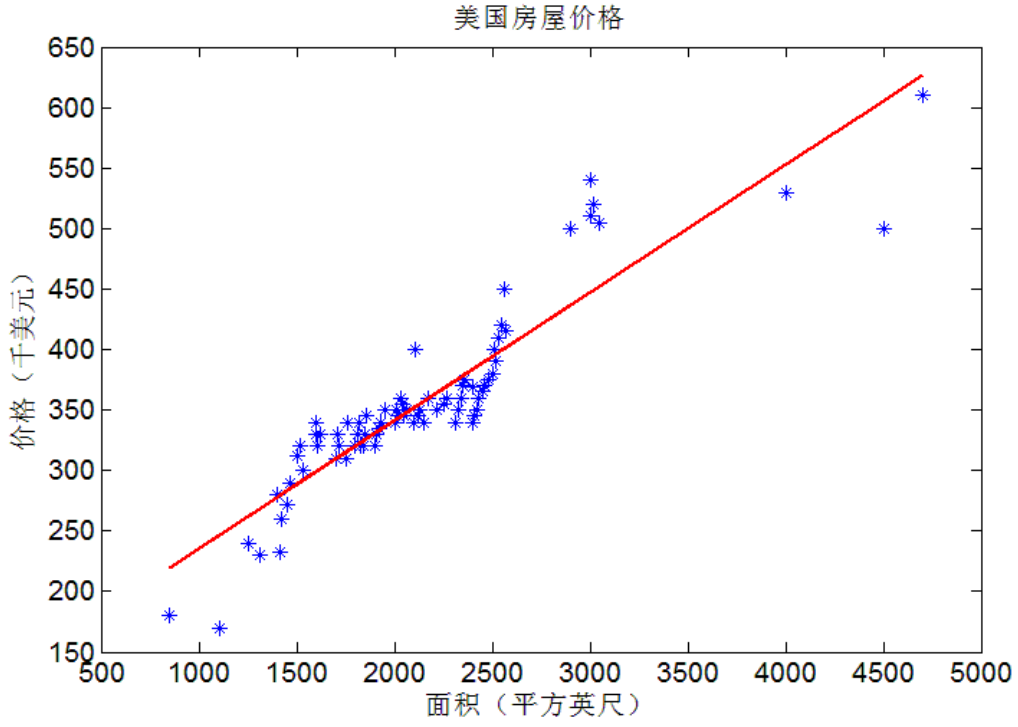
第五课—线性回归/逻辑回归算法

目录

- 数学模型
- 梯度下降算法
- 正则化
- ML回归算法参数详解
- ML实例
- 在线广告点击预估

数学模型





一元线性回归 $h_\theta(x) = \theta_0 + \theta_1 x$

多元线性回归 $h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T X$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\min_\theta J_\theta$$

批量梯度下降算法

损失函数 $$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$$

J(θ)的极小值问题 ⟹ 梯度下降法

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

α为学习速率

样本数量*m*为1时

$$\frac{\partial}{\partial\theta_j}J(\theta)=\frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x)-y)^2$$

$$=2\cdot\frac{1}{2}(h_\theta(x)-y)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x)-y)$$

$$=(h_\theta(x)-y)\cdot\frac{\partial}{\partial\theta_j}(\sum_{i=0}^{n}\theta_i x_i-y)$$

$$=(h_\theta(x)-y)x_j$$

$$\theta_j := \theta_j+\alpha(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

样本数量*m*不为1时

$$\theta_j := \theta_j+\alpha\sum_{i=1}^{m}(y^{(i)}-h_\theta(x^{(i)}))x_j^{(i)}$$

随机梯度下降算法

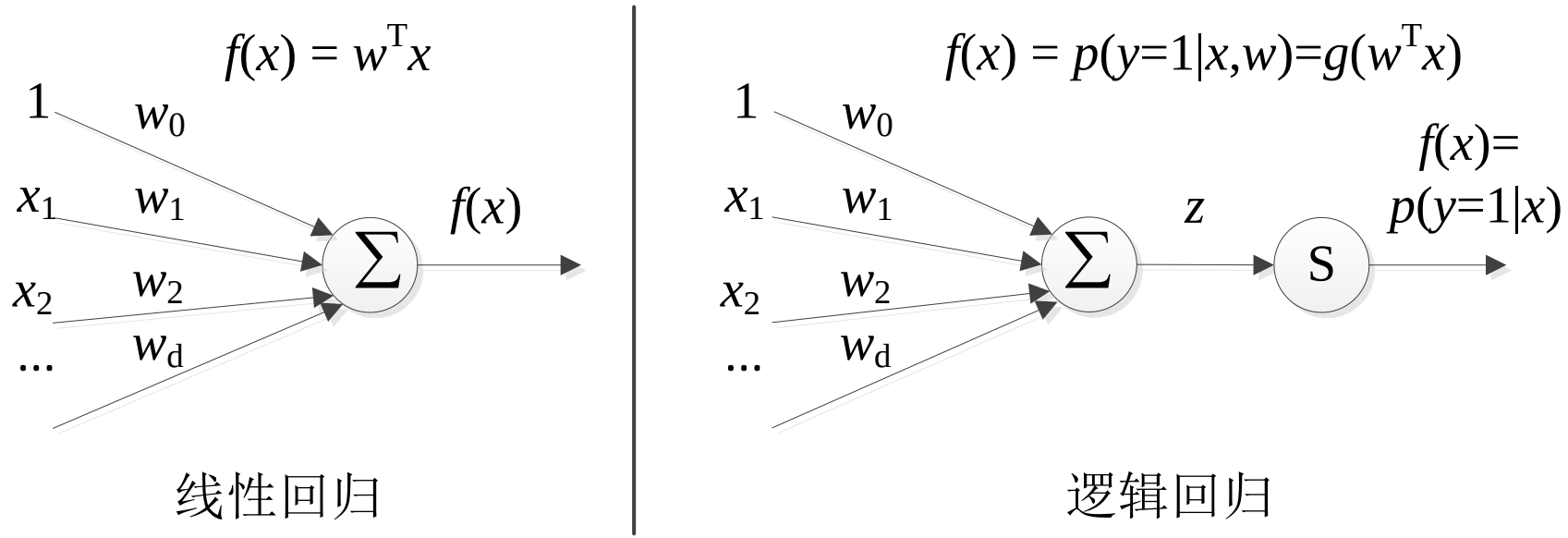
- 当样本集数据量m很大时，批量梯度下降算法每迭代一次的复杂度为$O(mn)$，复杂度很高。

```
Loop{
    For  i=1 to m{
        θ_j := θ_j + α(y^(i) − h_θ(x^(i)))x_j^(i)      (对于每个参数 j)
    }
}
```

- 即每读取一条样本，就迭代对 进行更新，这样迭代一次的算法复杂度为$O(n)$。

数学模型

- 线性回归与逻辑回归对比



数学模型

逻辑回归与线性回归的不同点在于：将线性回归的输出范围，例如从负无穷到正无穷，压缩到 0 和 1 之间；把大值压缩到这个范围还有个很好的用处，就是可以消除特别冒尖的变量的影响。

Logistic 函数（或称为 Sigmoid 函数），函数形式为：

$$g(z)=\frac{1}{1+e^{-z}}$$

Sigmoid 函数有个很漂亮的“S”形，如图 5-2 所示。

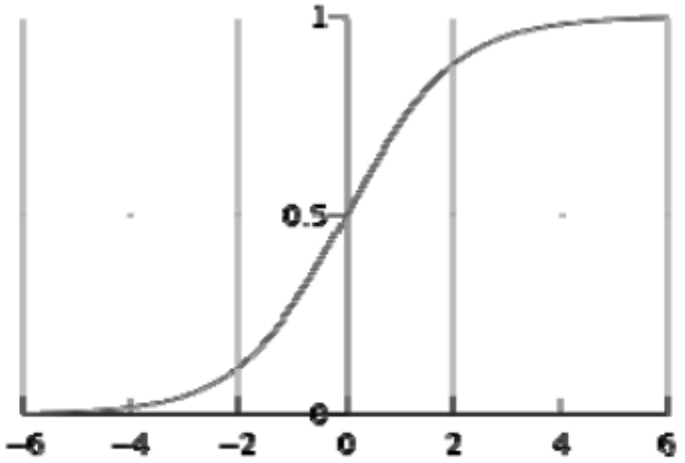


图 5-2　Sigmoid 函数

数学模型

给定 n 个特征 $x=(x_1,x_2,...,x_n)$，设条件概率 $p(y=1|x)$为观测样本 y 相对于事件因素 x 发生的概率，用 Sigmoid 函数表示为：

$$p(y=1|x)=\pi(x)=\frac{1}{1+e^{-g(x)}}$$

其中，$g(x)=w_0+w_1x_1+...+w_nx_n$，

那么在 x 条件下 y 不发生的概率为：

$$p(y=0|x)=1-p(y=1|x)=\frac{1}{1+e^{g(x)}}$$

假设现在有 m 个相互独立的观测事件 $y=(y^{(1)},y^{(2)},...,y^{(m)})$，则一个事件 $y^{(i)}$ 发生的概率为（$y^{(i)}=1$）：

$$p(y^{(i)})=p^{y^{(i)}}(1-p)^{1-y^{(i)}}$$

当 $y^{(i)}=1$ 的时候，后面那一项没有了，那就只剩下 x 属于 y=1 的概率；当 $y^{(i)}=0$ 的时候，第一项没有了，那就只剩下后面那个 x 属于 y=0 的概率（1 减去 x 属于 1 的概率）。所以不管 $y^{(i)}$ 是 0 还是 1，上面得到的数，都是(x，y)出现的概率。那我们的整个样本集，也就是 m 个独立样本出现的似然函数为（因为每个样本都是独立的，所以 m 个样本出现的概率就是它们各自出现的概率相乘）：

$$L(\theta)=\prod_{i=1}^{m}f(x;\theta)=\prod_{i=1}^{m}(\pi(x))^{y^{(i)}}(1-\pi(x))^{1-y^{(i)}}$$

数学模型

然后我们的目标是求出使这一似然函数的值最大的参数估计，最大似然估计就是求出参数 $\theta_0,\theta_1,...,\theta_n$，使得 $L(\theta)$取得最大值，对函数 $L(\theta)$取对数得到：

$$
\begin{aligned}
L(\theta) &= \log(\prod p(y^{(i)}=1|x^{(i)})^{y^{(i)}}(1-p(y^{(i)}=1|x^{(i)}))^{1-y^{(i)}}) \\
&= \sum_{i=1}^{m} y^{(i)}\log p(y^{(i)}=1|x^{(i)}) + (1-y^{(i)})\log(1-p(y^{(i)}=1|x^{(i)})) \\
&= \sum_{i=1}^{m} y^{(i)}\log\frac{p(y^{(i)}=1|x^{(i)})}{1-p(y^{(i)}=1|x^{(i)})} + \sum_{i=1}^{m}\log(1-p(y^{(i)}=1|x^{(i)})) \\
&= \sum_{i=1}^{m} y^{(i)}(\theta_0+\theta_1 x_1^{(i)}+...+\theta_n x_n^{(i)}) + \sum_{i=1}^{m}\log(1-p(y^{(i)}=1|x^{(i)})) \\
&= \sum_{i=1}^{m} y^{(i)}(\theta^{\mathrm{T}}x^{(i)}) - \sum_{i=1}^{m}\log(1+e^{\theta^{\mathrm{T}}x^{(i)}})
\end{aligned}
$$

最大似然估计就是求使 $L(\theta)$取最大值时的 θ，其实这里可以使用梯度上升法求解，求得的 θ 就是要求的最佳参数。我们也可以乘一个负的系数$-1/m$，转换成梯度下降法求解，$L(\theta)$转换成 $J(\theta)$：

$$J(\theta) = -\frac{1}{m}L(\theta)$$

所以，取 $J(\theta)$最小值时的 θ 为要求的最佳参数。如何调整 θ 以使得 $J(\theta)$取得最小值有很多方法，通常采用梯度下降法。

梯度下降算法

$$\sum_{i=1}^{m} y^{(i)}(\theta^{\mathrm{T}} x^{(i)}) - \sum_{i=1}^{m} \log(1 + e^{\theta^{\mathrm{T}} x^{(i)}})$$

θ 更新过程：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial_{\theta_j}} J(\theta)$$

$$\frac{\partial}{\partial_{\theta_j}} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} x_j^{(i)} - g(\theta^{\mathrm{T}} x^{(i)}) x_j^{(i)})$$

$$= \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

θ 更新过程可以写成：

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

上式 Σ(...)是一个求和的过程，需要循环 m 次，可以转换成如下向量化的过程。

梯度下降算法

约定训练数据的矩阵形式如下，x 的每一行为一条训练样本，而每一列为不同的特征值：

$$x=\begin{bmatrix}x_1\\ \dots\\ x_m\end{bmatrix}=\begin{bmatrix}x_{11} & \dots & x_{1n}\\ \dots & \dots & \dots\\ x_{m1} & \dots & x_{mn}\end{bmatrix},\quad y=\begin{bmatrix}y_1\\ \dots\\ y_m\end{bmatrix},\quad \theta=\begin{bmatrix}\theta_1\\ \dots\\ \theta_m\end{bmatrix}$$

$$A=x\cdot\theta=\begin{bmatrix}x_{10} & \dots & x_{1n}\\ \dots & \dots & \dots\\ x_{m1} & \dots & x_{mn}\end{bmatrix}\cdot\begin{bmatrix}\theta_0\\ \dots\\ \theta_m\end{bmatrix}=\begin{bmatrix}\theta_0x_{10}+\theta_1x_{11}+\dots+\theta_nx_{1n}\\ \dots\\ \theta_0x_{m0}+\theta_1x_{m1}+\dots+\theta_nx_{mn}\end{bmatrix}$$

$$E=h_\theta(x)-y=\begin{bmatrix}g(A_1)-y_1\\ \dots\\ g(A_m)-y_m\end{bmatrix}=\begin{bmatrix}e_1\\ \dots\\ e_m\end{bmatrix}=g(A)-y$$

综上所述，θ 更新的步骤如下：

1） $A=x\cdot\theta$

2） $E=g(A)-y$

3） $\theta:=\theta-\alpha x^{\mathrm{T}}E$

$g(A)$的参数 A 为一列向量，所以实现 g 函数时要支持列向量作为参数，并返回列向量。由上式可知 $h_\theta(x)-y$ 可由 $g(A)-y$ 计算求得。

θ 更新过程可以改为：

$$\theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}=\theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}e^{(i)}x_j^{(i)}=\theta_j-\alpha\frac{1}{m}x^{\mathrm{T}}E$$

正则化

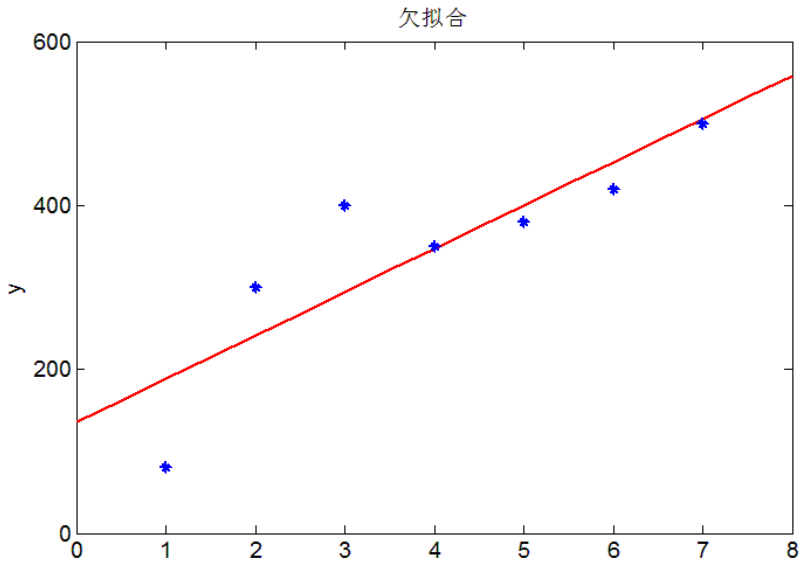


图 5-3 $y = \theta_0 + \theta_1 x$

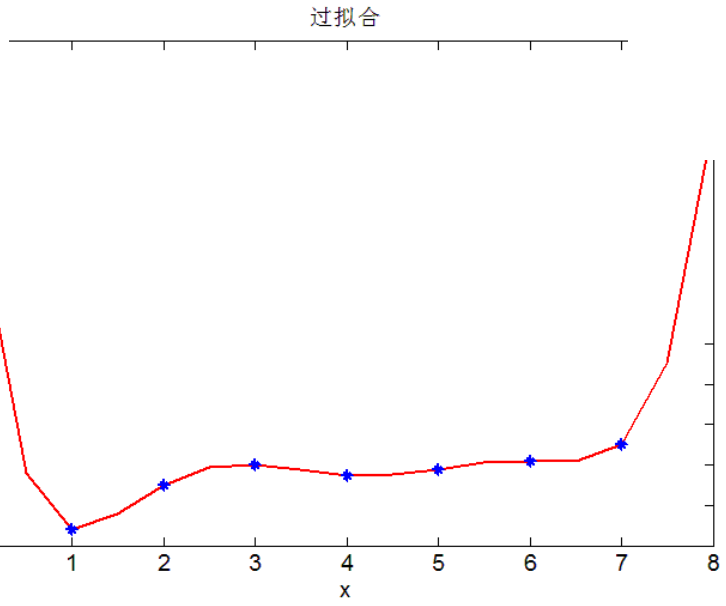


图 5-5 $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5 + \theta_6 x^6$

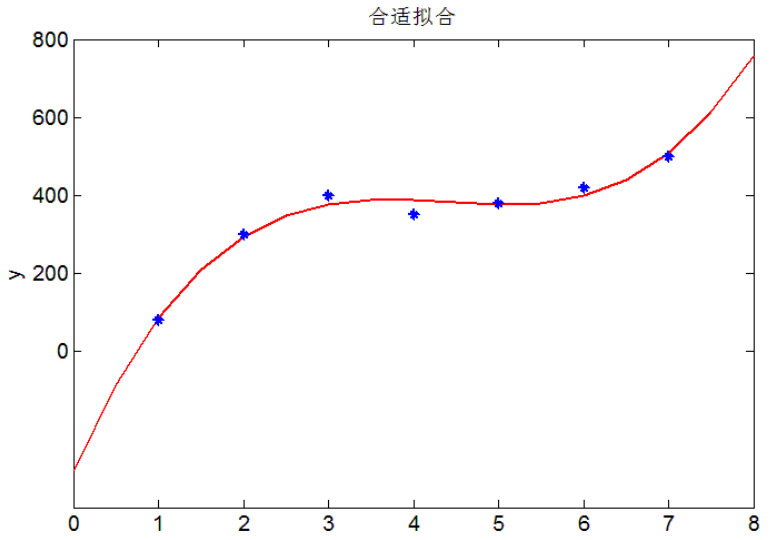


图 5-4 $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

正则化

2）正则化（特征较多时比较有效）。

保留所有特征，但减少 θ 的大小。正则化是结构风险最小化策略的实现，是在经验风险上加一个正则化项或惩罚项。正则化项一般是模型复杂度的单调递增函数，模型越复杂，正则化项就越大。

从直观上来看，如果我们想解决这个例子中的过拟合问题，最好能将 x^4 、 x^5 、 x^6 的影响消除，也就是让 θ_4 、 θ_5 、 θ_6 约等于 0。假设我们对 θ_4 、 θ_5 、 θ_6 进行惩罚，并且令其很小，一个简单的办法就是给原有的 Cost 函数加上略大惩罚项，例如：

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 + 1000\theta_4^2 + 1000\theta_5^2 + 1000\theta_6^2$$

这样在最小化 Cost 函数的时候， θ_4 、 θ_5 、 θ_6 约等于 0。

正则化

损失函数如下：

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i)-y_i)^2+\lambda\sum_{j=1}^{n}\theta_j^2$$

在损失函数里加入一个正则化项，正则化项就是权重的 L1 或者 L2 范数乘以一个正则系数，用来控制损失函数和正则化项的比重。直观地理解，首先防止过拟合的目的就是防止最后训练出来的模型过分地依赖某一个特征。当最小化损失函数的时候，某一维度很大，拟合出来的函数值与真实值之间的差距很小，通过正则化可以使整体的损失值变大，从而避免了过分依赖某一维度的结果。当然，加正则化的前提是特征值要进行归一化。

正则化

- ElasticNet被定义为L1和L2正则化项的凸组合：

$$\alpha(\lambda\|\mathbf{w}\|_1) + (1-\alpha)\left(\frac{\lambda}{2}\|\mathbf{w}\|_2^2\right), \alpha \in [0,1], \lambda \geq 0$$

随机梯度下降算法

2. 随机梯度下降算法

当样本集数据量 m 很大时，批量梯度下降算法每迭代一次的复杂度为 $O(mn)$，复杂度很高。因此，为了减少复杂度，当 m 很大时，我们更多时候使用随机梯度下降算法（stochastic gradient descent），算法如下所示：

```
Loop{
    For  i=1 to m{
        θ_j := θ_j + α(y^(i) − h_θ(x^(i)))x_j^(i)      (对于每个参数 j)
    }
}
```

即每读取一条样本，就迭代对 θ^T 进行更新。然后判断其是否收敛，若没收敛，则继续读取样本进行处理。如果所有样本都读取完毕了，则循环重新从头开始读取样本进行处理。

这样迭代一次的算法复杂度为 $O(n)$。对于大数据集，很有可能只需读取一小部分数据，函数 $J(\theta)$就收敛了。比如样本集数据量为 100 万，则有可能读取几千条或几万条时，函数就达到了收敛值。所以当数据量很大时，更倾向于选择随机梯度下降算法。

Spark 2.0 ML 讲解

线性回归

ML训练

参数	说明
setElasticNetParam(value: Double)	setElasticNetParam=0.0 为L2正则化 setElasticNetParam=1.0 为L1正则化 setElasticNetParam=(0,1.0) 为L1、L2组合 默认为0
setFeaturesCol(value: String)	指定特征列
setFitIntercept(value: Boolean)	是否需要偏置。 默认值为true
setLabelCol(value: String)	指定标签列
setMaxIter(value: Int)	最大迭代次数
setPredictionCol	指定预测列
setRegParam(value: Double)	设定正则因子，默认为0
setSolver(value: String)	设置用于优化求解器。 线性回归，支持l-bfgs（有限内存拟牛顿法）、normal（加权最小二乘法）和auto（自动选择）。
setStandardization(value: Boolean)	模型训练时是否将特征进行标准化处理，默认为true
setTol(value: Double)	设置迭代的收敛公差。 值越小准确性更高但迭代成本增加。 默认值为1E-6。
setWeightCol(value: String)	设置实例的权重值，如果不设置或者为空，默认所有实例的权重为1
setAggregationDepth(value: Int)	reeAggregate的建议深度（大于或等于2）。 默认值为2。如果特征维度较大或者数据的分区量大的时候，可以把该值调大。

实例代码

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.regression.{ LinearRegression, LinearRegressionModel }
import org.apache.spark.ml.{ Pipeline, PipelineModel }
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{ Vector, Vectors }
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{Vector, Vectors}
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

实例代码

```
import spark.implicits._

//1 训练样本准备
val training = spark.read.format("libsvm").load("hdfs:/[redacted]/sample_linear_regression_data.txt")
training.show(false)
```

```
import spark.implicits._
training: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+--------------------+-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
|label               |features
                                                                       |
+--------------------+-------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
|-9.490009878824548  |(10,[0,1,2,3,4,5,6,7,8,9],[0.4551273600657362,0.36644694351969087,-0.38256108933468047,-0.4458430198517267,0.331097
73887,-0.4485038611165952 4,-0.07269284838169332,0.5658035575800715])    |
|0.2577820163584905  |(10,[0,1,2,3,4,5,6,7,8,9],[0.8386555657374337,-0.1270180511534269,0.49981236251089 5,-0.2268662512813026 7,-0.645243
3358,0.6519317437756 42,-0.6555641246242951,0.17485476357259122])       |
|-4.438869807456516  |(10,[0,1,2,3,4,5,6,7,8,9],[0.5025608135349202,0.14208069682973434,0.16004976900412138,0.505019897181302,-0.9371635
86,-0.16462490649416 25,0.9480713629917628,0.4268125156464581 7])       |
|-19.782762789614537|(10,[0,1,2,3,4,5,6,7,8,9],[-0.0388509668871313,-0.41668700517639 18,0.8997202693189332,0.640983646772693 3,0.273289
298,-0.13067782971877 94,-0.08536581110461 15,-0.05462315824828923])     |
|-7.966593841555266  |(10,[0,1,2,3,4,5,6,7,8,9],[-0.06195495876886281,0.6546448480299902,-0.6979368909424835,0.6677324708883314,-0.079387
153688,-0.64145311825 01653,0.7313735926547045,-0.02681867634761192 5])   |
```

实例代码

```
//2 建立逻辑回归模型
val lr = new LinearRegression()
  .setMaxIter(500)
  .setRegParam(0.3)
  .setElasticNetParam(0.8)

//2 根据训练样本进行模型训练
val lrModel = lr.fit(training)
```

```
lr: org.apache.spark.ml.regression.LinearRegression = linReg_84930fa5687d
lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_84930fa5687d
```

Took 2 sec. Last updated by sunbowhuang at October 30 2017, 9:00:06 PM.

```
//2 打印模型信息
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
println(s"Intercept: ${lrModel.intercept}")
```

```
Coefficients: [0.0,0.32292516677405936,-0.3438548034562218,1.9156017023458414,0.05288058680386263,0.76596272045
  Intercept: 0.1598936844239736
Intercept: 0.1598936844239736
```

实例代码

```
val test = spark.createDataFrame(Seq(
  (5.601801561245534, Vectors.sparse(10, Array(0,1,2,3,4,5,6,7,8,9), Array(0.694918973
      .6676656789571533,-0.03553655732400762,0.14550349954571096,0.034600542078191854,
  (0.2577820163584905, Vectors.sparse(10, Array(0,1,2,3,4,5,6,7,8,9), Array(0.83865556
      .18869982177936828,-0.5804648622673358,0.651931743775642,-0.6555641246242951,0.1
  (1.5299675726687754, Vectors.sparse(10, Array(0,1,2,3,4,5,6,7,8,9), Array(-0.1307929
      .6361110540492223,0.7675261182370992,-0.2543488202081907,0.2927051050236915,0.68
test.show
```

```
test: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+------------------+--------------------+
|             label|            features|
+------------------+--------------------+
| 5.601801561245534|(10,[0,1,2,3,4,5,...|
|0.2577820163584905|(10,[0,1,2,3,4,5,...|
|1.5299675726687754|(10,[0,1,2,3,4,5,...|
+------------------+--------------------+
```

实例代码

```
//5 对模型进行测试
val test_predict = lrModel.transform(test)
test_predict.show
test_predict.select("features", "label", "prediction").collect().foreach {
  case Row(features: Vector, label: Double, prediction: Double) =>
    println(s"($features, $label) -> prediction=$prediction")
}
```

```
test_predict: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 1 more field]
+------------------+--------------------+--------------------+
|             label|            features|          prediction|
+------------------+--------------------+--------------------+
| 5.601801561245534|(10,[0,1,2,3,4,5,...| -2.0446543261749612|
|0.2577820163584905|(10,[0,1,2,3,4,5,...|-0.29559741764686487|
|1.5299675726687754|(10,[0,1,2,3,4,5,...|  1.6330567770307318|
+------------------+--------------------+--------------------+
((10,[0,1,2,3,4,5,6,7,8,9],[0.6949189734965766,-0.32697929564739403,-0.15359663581829275,-0.895186509052043
571096,0.034600542078191854,0.4223352065067103]), 5.601801561245534) -> prediction=-2.0446543261749612
((10,[0,1,2,3,4,5,6,7,8,9],[0.8386555657374337,-0.1270180511534269,0.499812362510895,-0.22686625128130267,-
2,-0.6555641246242951,0.17485476357259122]), 0.2577820163584905) -> prediction=-0.29559741764686487
((10,[0,1,2,3,4,5,6,7,8,9],[-0.13079299081883855,0.0983382230287082,0.15347083875928424,0.45507300685816965
07,0.2927051050236915,0.680182444769418]), 1.5299675726687754) -> prediction=1.6330567770307318
```

实例代码

```
    //6 模型摘要
    val trainingSummary = lrModel.summary

    // 每次迭代目标值
    val objectiveHistory = trainingSummary.objectiveHistory
    println(s"numIterations: ${trainingSummary.totalIterations}")
    println(s"objectiveHistory: [${trainingSummary.objectiveHistory.mkString(",")}]")
    trainingSummary.residuals.show()
    println(s"RMSE: ${trainingSummary.rootMeanSquaredError}")
    println(s"r2: ${trainingSummary.r2}")
```

```
trainingSummary: org.apache.spark.ml.regression.LinearRegressionTrainingSummary = org.apache.spark.m
objectiveHistory: Array[Double] = Array(0.49999999999999994, 0.4967620357443381, 0.4936361664340463,
numIterations: 7
objectiveHistory: [0.49999999999999994,0.4967620357443381,0.4936361664340463,0.4936351537897608,0.49
+--------------------+
|           residuals|
+--------------------+
|  -9.889232683103197|
|  0.5533794340053554|
|  -5.204019455758823|
| -20.566686715507508|
|    -9.4497405180564|
|  -6.909112502719486|
|  -10.00431602969873|
```

实例代码

```
    //7 模型保存与加载
    lrModel.save("hdfs://[redacted]/mlv2/lrmodel2")
    val load_lrModel = LinearRegressionModel.load("hdfs:[redacted]/mlv2/lrmodel2")
```

```
load_lrModel: org.apache.spark.ml.regression.LinearRegressionModel = linReg_84930fa5687d
```

Spark 2.0 ML 讲解

逻辑回归

ML训练参数

参数	说明
setElasticNetParam(value: Double)	setElasticNetParam=0.0 为L2正则化 setElasticNetParam=1.0 为L1正则化 setElasticNetParam=(0,1.0) 为L1、L2组合 默认为0
setFamily(value: String)	auto” ：根据类的数量自动选择系列：如果numClasses == 1 || numClasses == 2，设为“二项式”。 否则，设置为“多项式” "binomial"：二元逻辑回归。 "multinomial"：多元Logistic（softmax）回归。 默认为 "auto"
setFeaturesCol(value: String)	指定特征列
setFitIntercept(value: Boolean)	是否需要偏置。 默认值为true
setLabelCol(value: String)	指定标签列
setMaxIter(value: Int)	最大迭代次数
setPredictionCol	指定预测列
setProbabilityCol	指定预测概率值列
setRawPredictionCol	指定原始预测列
setRegParam(value: Double)	设定正则因子，默认为0
setStandardization(value: Boolean)	模型训练时是否将特征进行标准化处理，默认为true

ML训练参数

setThreshold(value: Double)	二分类阈值[0-1]，默认为0.5，如果预测值大于0.5则为1，否则为0
setThresholds(value: Array[Double])	多元分类阈值[0-1]，默认为0.5
setTol(value: Double)	设置迭代收敛公差目标值，默认为1E-6
setWeightCol(value: String)	设置实例的权重值，如果不设置或者为空，默认所有实例的权重为1
setAggregationDepth(value: Int)	reeAggregate的建议深度（大于或等于2）。 默认值为2。
setLowerBoundsOnCoefficients(value: Matrix)	在有限约束优化下拟合，则设置系数的下限。矩阵大小（类的数量，特征数）
setLowerBoundsOnIntercepts(value: Vector)	在有约束的优化下进行拟合，则设置偏置的下限。向量大小：类的数量
setUpperBoundsOnCoefficients(value: Matrix)	在有限约束优化下拟合，则设置系数的上限。矩阵大小（类的数量，特征数）
setUpperBoundsOnIntercepts(value: Vector)	在有约束的优化下进行拟合，则设置偏置的上限。向量大小：类的数量

实例代码

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.classification.{ BinaryLogisticRegressionSummary, LogisticRegression, LogisticRegressionModel }
import org.apache.spark.ml.{ Pipeline, PipelineModel }
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{ Vector, Vectors }
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.ml.feature._
import org.apache.spark.ml.classification.{BinaryLogisticRegressionSummary, LogisticRegression, LogisticRegressionModel}
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.param.ParamMap
import org.apache.spark.ml.linalg.{Vector, Vectors}
import org.apache.spark.sql.catalyst.encoders.ExpressionEncoder
import org.apache.spark.sql.Encoder
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql._
import org.apache.spark.sql.SparkSession
```

实例代码

```
import spark.implicits._

//1 训练样本准备
val training = spark.read.format("libsvm").load("hdfs://[redacted]/sample_libsvm_data.txt")
training.show
```

```
|  1.0|(692,[151,152,153...|
|  0.0|(692,[129,130,131...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[99,100,101,...|
|  0.0|(692,[154,155,156...|
|  0.0|(692,[127,128,129...|
|  1.0|(692,[154,155,156...|
|  0.0|(692,[153,154,155...|
|  0.0|(692,[151,152,153...|
|  1.0|(692,[129,130,131...|
|  0.0|(692,[154,155,156...|
|  1.0|(692,[150,151,152...|
|  0.0|(692,[124,125,126...|
|  0.0|(692,[152,153,154...|
|  1.0|(692,[97,98,99,12...|
|  1.0|(692,[124,125,126...|
+-----+--------------------+
only showing top 20 rows
```

实例代码

```
//2 建立逻辑回归模型
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)

//2 根据训练样本进行模型训练
val lrModel = lr.fit(training)

//2 打印模型信息
println(s"Coefficients: ${lrModel.coefficients} Intercept: ${lrModel.intercept}")
println(s"Intercept: ${lrModel.intercept}")
```

```
lr: org.apache.spark.ml.classification.LogisticRegression = logreg_1b8e36eea968
lrModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_1b8e36eea968
Coefficients: (692,[244,263,272,300,301,328,350,351,378,379,405,406,407,428,433,434,455,456,461,462,483,484,4
04298E-4,-2.0300642473486668E-4,-3.1476183314863995E-5,-6.842977602660743E-5,1.5883626898239883E-5,1.40234970
E-4,2.840248179122762E-4,-1.1541084736508837E-4,3.85996886312906E-4,6.35019557424107E-4,-1.1506412384575676E-
075579290126E-4,2.739010341160883E-4,2.7730456244968115E-4,-9.838027027269332E-5,-3.808522443517704E-4,-2.531
11331293E-4]) Intercept: 0.22456315961250325
Intercept: 0.22456315961250325
```

实例代码

```
//3 建立多元回归模型
val mlr = new LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8).setFamily("multinomial")

//3 根据训练样本进行模型训练
val mlrModel = mlr.fit(training)

//3 打印模型信息
println(s"Multinomial coefficients: ${mlrModel.coefficientMatrix}")
println(s"Multinomial intercepts: ${mlrModel.interceptVector}")
```

```
mlr: org.apache.spark.ml.classification.LogisticRegression = logreg_6d89f54c4645
mlrModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_6d89f54c4645
Multinomial coefficients: 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ... (692 total)
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...
Multinomial intercepts: [-0.12065879445860686,0.12065879445860686]
```

实例代码

```
//4 测试样本
val test = spark.createDataFrame(Seq(
  (1.0, Vectors.sparse(692, Array(10, 20, 30), Array(-1.0, 1.5, 1.3))),
  (0.0, Vectors.sparse(692, Array(45, 175, 500), Array(-1.0, 1.5, 1.3))),
  (1.0, Vectors.sparse(692, Array(100, 200, 300), Array(-1.0, 1.5, 1.3))))).toDF("label", "features")
test.show
```

```
test: org.apache.spark.sql.DataFrame = [label: double, features: vector]
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  1.0|(692,[10,20,30],[...|
|  0.0|(692,[45,175,500]...|
|  1.0|(692,[100,200,300...|
+-----+--------------------+
```

实例代码

```
//5 对模型进行测试
val test_predict = lrModel.transform(test)
test_predict.show
test_predict.select("features", "label", "probability", "prediction").collect().foreach {
  case Row(features: Vector, label: Double, prob: Vector, prediction: Double) =>
    println(s"($features, $label) -> prob=$prob, prediction=$prediction")
}
```

```
test_predict: org.apache.spark.sql.DataFrame = [label: double, features: vector ... 3 more fields]
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  1.0|(692,[10,20,30],[...|[-0.2245631596125...|[0.44409395157819...|       1.0|
|  0.0|(692,[45,175,500]...|[-0.2245631596125...|[0.44409395157819...|       1.0|
|  1.0|(692,[100,200,300...|[-0.2242992512603...|[0.44415910478520...|       1.0|
+-----+--------------------+--------------------+--------------------+----------+
((692,[10,20,30],[-1.0,1.5,1.3]), 1.0) -> prob=[0.4440939515781922,0.5559060484218078], prediction=1.0
((692,[45,175,500],[-1.0,1.5,1.3]), 0.0) -> prob=[0.4440939515781922,0.5559060484218078], prediction=1.0
((692,[100,200,300],[-1.0,1.5,1.3]), 1.0) -> prob=[0.44415910478520204,0.555840895214798], prediction=1.0
```

实例代码

```
//6 模型摘要
val trainingSummary = lrModel.summary

//6 每次迭代目标值
val objectiveHistory = trainingSummary.objectiveHistory
println("objectiveHistory:")
objectiveHistory.foreach(loss => println(loss))
```

```
trainingSummary: org.apache.spark.ml.classification.LogisticRegressionTrainingSummary = org.apache.spark.ml.class
objectiveHistory: Array[Double] = Array(0.6833149135741672, 0.6662875751473734, 0.6217068546034618, 0.61272652458
06089243339022, 0.5894724576491042, 0.5882187775729587)
objectiveHistory:
0.6833149135741672
0.6662875751473734
0.6217068546034618
0.6127265245887887
0.6060347986802873
0.6031750687571562
0.5969621534836274
0.5940743031983118
0.5906089243339022
0.5894724576491042
0.5882187775729587
```

实例代码

```
//6 计算模型指标数据
val binarySummary = trainingSummary.asInstanceOf[BinaryLogisticRegressionSummary]

//6 AUC指标
val roc = binarySummary.roc
roc.show()
val AUC = binarySummary.areaUnderROC
println(s"areaUnderROC: ${binarySummary.areaUnderROC}")
```

```
|FPR|                  TPR|
+---+--------------------+
|0.0|                 0.0|
|0.0|0.017543859649122806|
|0.0| 0.03508771929824561|
|0.0| 0.05263157894736842|
|0.0| 0.07017543859649122|
|0.0| 0.08771929824561403|
|0.0| 0.10526315789473684|
|0.0| 0.12280701754385964|
|0.0| 0.14035087719298245|
|0.0| 0.15789473684210525|
|0.0| 0.17543859649122806|
|0.0| 0.19298245614035087|
|0.0| 0.21052631578947367|
|0.0| 0.22807017543859648|
|0.0| 0.24561403508771928|
```

实例代码

```
//6 设置模型阈值
//不同的阈值，计算不同的F1，然后通过最大的F1找出并重设模型的最佳阈值。
val fMeasure = binarySummary.fMeasureByThreshold
fMeasure.show
//获得最大的F1值
val maxFMeasure = fMeasure.select(max("F-Measure")).head().getDouble(0)
//找出最大F1值对应的阈值（最佳阈值）
val bestThreshold = fMeasure.where($"F-Measure" === maxFMeasure).select("threshold").head().getDouble(0)
//并将模型的Threshold设置为选择出来的最佳分类阈值
lrModel.setThreshold(bestThreshold)
```

```
fMeasure: org.apache.spark.sql.DataFrame = [threshold: double, F-Measure: double]
+------------------+--------------------+
|         threshold|           F-Measure|
+------------------+--------------------+
|0.7845860015371142|0.034482758620689655|
|0.7843193344168922| 0.06779661016949151|
|0.7842976092510131|                 0.1|
|0.7842531051133191| 0.13114754098360656|
|0.7835792429453297| 0.16129032258064516|
|0.7835223585829078|  0.1904761904761905|
| 0.783284563364102|             0.21875|
|0.7832449070254992| 0.24615384615384614|
|0.7830630257264691|  0.2727272727272727|
|0.7783754276111222|  0.4799999999999999|
|0.7771658291080574|                 0.5|
|0.7769914303593917|  0.5194805194805194|
+------------------+--------------------+
only showing top 20 rows
maxFMeasure: Double = 1.0
bestThreshold: Double = 0.5585022394278357
res25: lrModel.type = logreg_1b8e36eea968
```

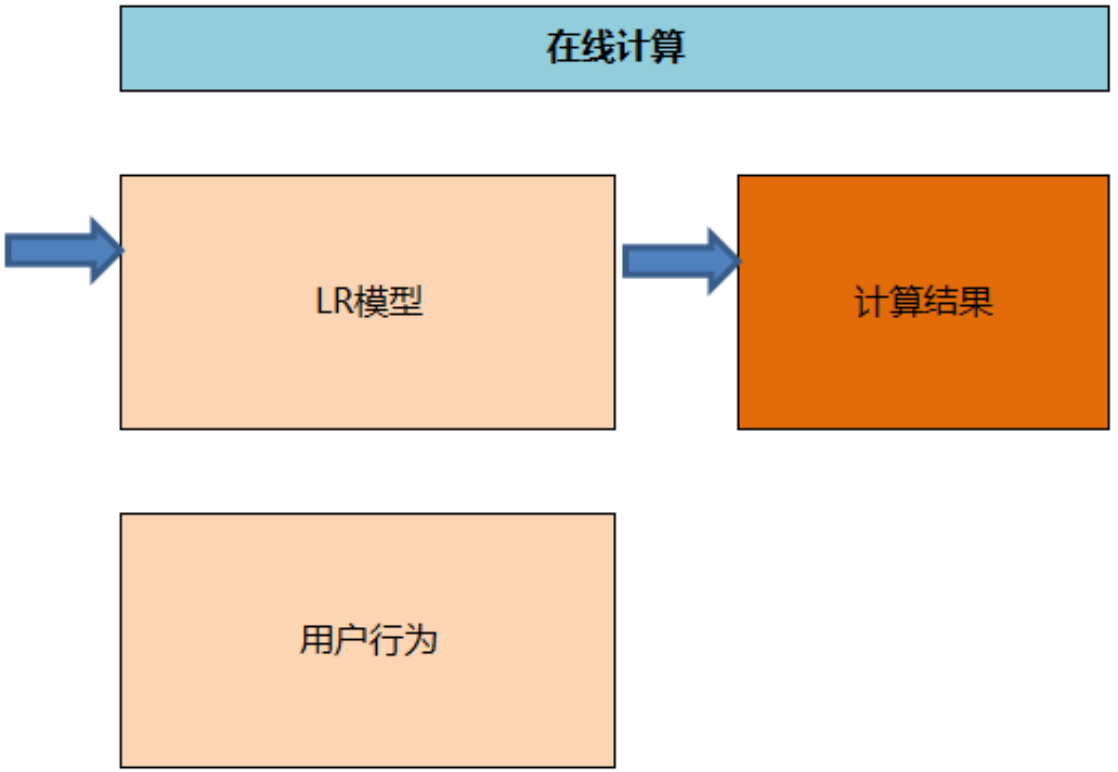
实例代码

```
    //7 模型保存与加载
    lrModel.save("hdfs://[redacted]lv2/lrmodel")
    val load_lrModel = LogisticRegressionModel.load("hdfs://[redacted]/mlv2/lrmodel")

load_lrModel: org.apache.spark.ml.classification.LogisticRegressionModel = logreg_f32b9f2b079b
```

在线广告点击预估

离线计算							
用户对广告行为				用户画像			广告点击
浏览	点击	收藏	.....	人口属性	爱好	.....	



Thanks

FAQ时间