平均随机梯度下降法

赵世瑜

2016 年 5 月 30 日

梯度下降（Gradient Descent，简称 GD）法作为一种最优化算法，是最小化经验风险函数或损失函数时常用的一种方法。随着梯度下降法的发展，目前梯度下降法已出现不同的迭代思路或形式，如批量梯度下降（GD）、随机梯度下降（SGD）、小批量随机梯度下降（Mini-batch Stochastic Gradient Descent，MSGD）及平均随机梯度下降（ASGD）法等。这里首先介绍一下各种梯度下降法的迭代思路，然后结合线性回归进行梯度下降法进行简单的应用。

1 GD

1.1 梯度下降法

标量场的梯度是一个矢量场。标量场中某点的梯度方向为标量场增加最快的方向，大小为改点的变化率。

若 $h(x)$ 是目标函数，$J(\theta)$ 为损失函数，θ 为参数，即需要迭代求解的值。参数 θ 的估计通过最小化损失函数 $J(\theta)$ 进行估计。

通过训练数据对参数 θ 进行学习，根据损失函数的梯度进行一次次迭代来更新参数 θ，过程如式（1）所示：

$$\theta_{j+1} = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta J(\theta_i) \qquad (1)$$

由式(1)可知，通过第 j 次的参数 θ_j 和损失函数的梯度 $\nabla_\theta J(\theta_i)$ 对第 $j+1$ 次的参数 θ_{j+1} 进行更新。其中 m 为训练样本数量。α 为学习速率，即每次更新参数的步长大小参数，学习速率 α 不宜过大，也不宜过小，过大会导致无法收敛或得不到较好的结果，过小导致学习速度慢等问题，一种简单的思想是随着结果接近全局最优时，也不断减小学习速率 α，使结果尽量逼近全局最优。

梯度下降法最原始的形式为批量梯度下降（BGD）法，批量梯度下降法的思路是在更新每一参数时都使用所有的样本来进行更新。批量梯度下降具有能获得全局最优解，易于并行实现等，但当训练样本的数目很大时，训练过程将变得很慢。

1.2 GD 在线性回归中的应用

假设线性回归模型为$h_\theta(x)$， $h_\theta(x)$如式（2）所示：

$$h_\theta(x)=\sum_{k=0}^{\mathrm{K}}\theta_k x_k \tag{2}$$

其中，一般情况下x_0=1，那么其对应的损失函数为：

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2 \tag{3}$$

其中分母上的 2 只是为了求偏导后约去指数 2，对损失函数求偏导得：

$$\frac{\partial J(\theta)}{\theta_k}=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x^i \tag{4}$$

由于是最小化经验风险函数，因此按照参数θ的负梯度方向对参数进行更新。参数更新过程如式（5）所示：

$$\theta_{(j+1)k}=\theta_{jk}-\alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x^i \tag{5}$$

即用所有的样本更新一次参数θ。实现见附录代码，具体实现时使用批量梯度的思想，将代码中的 batchSize 大小设为训练样本数目，即为批量梯度下降法。

2 SGD

2.1 随机梯度下降法

由于批量梯度下降法在更新每一个参数时，都需要用到所有的训练样本，导致训练过程会随着样本数量的加大而变得异常的缓慢。随机梯度下降（SGD）法正好可以解决这一弊端。随机梯度下降法更新参数的策略是，每次迭代是从总体样本中随机选取一个样本对参数进行更新，其更新参数的过程如式（6）所示：

$$\theta_{j+1}=\theta_j-\alpha\nabla_\theta J(\theta_j) \tag{6}$$

随机梯度下降通过随机选取一个个样本来迭代更新一次参数θ，当样本量很大时，那么可能只用其中的一部分样本，就已经将θ迭代到最优解了。

相对于批量梯度下降，迭代一次需要用到所有的训练样本，一次迭代不可能最优，一般都需要多次对迭代。但是，随机梯度下降法伴随的一个问题是噪音比批量梯度下降法更多，使得随机梯度下降法并不是每次迭代都向着全局最优的方向进行。

相对于批量梯度下降法，随机梯度下降法具有训练速度快的优点，是存在的问题是得到的结果不一定是全局最优，很有可能是局部最优。即随机梯度下降法很有可能陷入局部最优。

2.2 小批量随机梯度下降法

批量梯度下降法和随机梯度下降法各有优缺点，针对批量梯度下降法和随机梯度下降法存在的问题，能不能寻找一种策略，既有批量梯度下降法的优点，也有随机梯度下降法的优点。也就是在批量梯度和随机梯度之间进行一种折衷，那么想法是不使用批量梯度下降中的所有样本更新参数，也不使用一个样本更新参数，而是从总体样本中随机选取一部样本更新参数，如 10 或 100 等。这就是小批量随机梯度下降法的思想。其更新参数 θ 的过程如式（7）所示：

$$\theta_{j+1} = \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta J(\theta_i) \tag{7}$$

式（7）中，n 为训练样本总量 m 的一个子集，$n \le m$，当然一般不会使用 $n = m$ 的情况，而是 n 远小于 m，所以叫小批量，而且这个小批量子集中的每一个样本都是从总体样本中随机选择的。

2.3 SGD 在线性回归中的应用

随机梯度下降法在线性回归中的应用，将公式（5）换为如下所示，即为随机梯度下降法；

$$\theta_{(j+1)k} = \theta_{jk} - \alpha(h_\theta(x^j) - y^j)x^j \tag{8}$$

如果是使用小批量随机梯度下降法，则如公式（9）所示:

$$\theta_{(j+1)k} = \theta_{jk} - \alpha \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^i) - y^i)x^i \tag{9}$$

其中 n 为每一批次训练样本的大小。

在具体的代码实现时，仍然使用分批次的思想，如果 batchSize 的大小设为 1，则对应随机梯度下降法，如果 batchSize 的大小大于 1，则为小批量随机梯度下降法。

随机策略使用两种方式实现，一种是对样本进行一轮一轮迭代，每一轮中的每一批次的样本是随机选取的，但每一轮中每一个样本都会被使用到，而且一轮只使用一次。另一种随机策略是不对所有训练样本进行一轮轮迭代，而是在每次更新参数时都从所有样本中随机选取一部分样本进行迭代，也就是连续两次更新参数是可能用到了同一个样本，但当迭代次数较多时，应该所有样本都已经使用过了。这两种迭代策略和样本的分布有关，后一种策略其实是假设样本服从均匀分布，这里不再详细说明。

3 ASGD

3.1 平均随机梯度下降法

在小批量随机梯度下降法中，使用 n 个小批量样本迭代一次更新参数 θ。那么，另外一

种策略是，在n个小批量样本总仍然对每一个样本进行一次参数更新，但是在更新当前批次的参数θ是，用上一批次中更新的参数θ的平均值来更新当前批次中的参数，即$\theta_j = \frac{1}{n}\sum_{i=1}^{n}\theta_{ji}$，其对应的参数更新过程如式（10）或（11）所示：

$$\theta_{j+1} = \frac{1}{n}\sum_{i=1}^{n}\theta_{ji} - \alpha \nabla_{\theta} J(\theta_j) \tag{10}$$

$$\theta_{j+1} = \theta_j - \alpha \nabla_{\theta} J(\theta_j) \tag{11}$$

由公式（10）可知，参数更新时，使用上一批次参数θ_j的平均值对当前批次参数θ_{j+1}进行更新。

3.2 ASGD 在线性回归中的应用

根据平均随机梯度下降法的原理，线性回归在平均随机梯度下降法中更新参数时，需要先计算前一批次中参数θ的平均值，即：$\theta_{jk} = \frac{1}{n}\sum_{i=1}^{n}\theta_{jik}$，参数的更新过程如公式（12）所示：

$$\theta_{(j+1)k} = \frac{1}{n}\sum_{i=1}^{n}\theta_{jik} - \alpha(h_{\theta}(x^j) - y^j)x^j \tag{12}$$

具体实现详见附录代码。

4 总结

主要介绍了梯度下降法及其不同的迭代形式，结合线性回归进行了简单应用，试验代码中的训练数据是根据线性方程随机生成的。

梯度下降法不同迭代形式在线性回归中的应用，实现使用 Java 语言。从简单的结果来看，ASGD 能够收敛到更好的结果，误差可以达到 10^{-7} 以下，而其他的迭代策略很难迭代达到这一误差水平。

参考文献

[1] L´eon Bottou. Stochastic Gradient Descent Tricks. Microsoft Research, Redmond, WA.

[2] L´eon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. NEC Labs America, Princeton NJ 08542, USA.

[3] http://leon.bottou.org/projects/sgd

[4] Francis Bach. Adaptivity of Averaged Stochastic Gradient Descent to Local Strong Convexity for Logistic Regression. Journal of Machine Learning Research 15 (2014) 367-399.

[5] http://m.blog.csdn.net/article/details?id=7950084

附录

梯度下降法不同迭代形式在线性回归中应用的 Java 实现

```
/**
 * 梯度下降法在线性回归中的应用
 *
 * @author Zhao Shiyu
 *
 */
public class ASGD {

    /**
     * 目标函数
     * @param theta 参数
     * @param x 变量
     * @return 结果
     * @throws Exception
     */
    public static double target(double[] theta, double[] x) throws Exception {
        if(theta.length != x.length) throw new Exception("维数不一致！");
        int len = theta.length;
        double ret = 0;
        for(int i = 0; i < len; i++) {
            ret += theta[i] * x[i];
        }
        return ret;
    }

    /**
     * 计算偏差
     * @param theta 参数
     * @param x 变量
     * @param y 真实结果
     * @return 偏差大小
     * @throws Exception
     */
    public static double dev(double[] theta, double[] x, double y) throws Exception {
        if(theta.length != x.length) throw new Exception("维数不一致！");
        return (target(theta, x) - y);
    }
```

```
/**
 * 计算多组数据偏差
 * @param theta
 * @param x
 * @param y
 * @return
 * @throws Exception
 */
public static double[] dev(double[] theta, double[][] x, double y[]) throws Exception {
	if(x.length != y.length) throw new Exception("维数不一致！ ");
	int len = y.length;
	double[] revRet = new double[len];
	for(int i = 0; i < len; i++) {
		revRet[i] = dev(theta, x[i], y[i]);
	}
	return revRet;
}

/**
 * 计算损失值
 * @param theta 参数集
 * @param x 变量集
 * @param y 真实结果集
 * @return 损失值大小
 * @throws Exception
 */
public static double loss(double[][] theta, double[][] x, double y[]) throws Exception {
	if(theta.length != x.length) throw new Exception("维数不一致！ ");
	int len = y.length;
	double lossSum = 0;
	for(int i = 0; i < len; i++) {
		lossSum +=    Math.pow(dev(theta[i], x[i], y[i]), 2.0);
	}
	return (lossSum / len);
}

/**
 * 计算损失值
 * @param theta
 * @param x
 * @param y
 * @return
 * @throws Exception
 */
```

```
    public static double loss(double[] theta, double[][] x, double y[]) throws Exception {
        if(theta.length != x[0].length) throw new Exception("维数不一致！");
        int len = y.length;
        double lossSum = 0;
        for(int i = 0; i < len; i++) {
            lossSum +=    Math.pow(dev(theta, x[i], y[i]), 2.0);
        }
        return (lossSum / len);
    }

    /**
     * 更新参数
     * @param theta 参数
     * @param x 变量
     * @param y 真实结果
     * @param alpha 学习速率
     * @return 更新后的参数
     * @throws Exception
     */
    public static double[] updateTheta(double[] theta, double[] x, double y, double alpha) throws Exception {
        if(theta.length != x.length) throw new Exception("维数不一致！");
        int len = theta.length;
        double[] newTheta = new double[len];

        double dev = dev(theta, x, y);
        double gradient = 0.0;
        for(int i = 0; i < len; i++) {
            for(int j = 0; j < len; j++) {
                if (i == 0) {//theta_0 计算稍微有点差别，这里到没差别，因为 batchX[i][0] 的值为 1
                    gradient += dev;
                } else {
                    gradient += dev * x[j];
                }
            }
            newTheta[i] =    (theta[i] - alpha * (gradient / len));
        }
        return newTheta;
    }

    public static double[] updateTheta(double[] theta, double[][] batchX, double[] batchY, double alpha) throws Exception {
        if(theta.length != batchX[0].length) throw new Exception("维数不一致！");
```

```
            int batchSize = batchX.length;
            int parameter = theta.length;
            double[][] batchTheta = new double[batchSize][parameter];
            for(int i = 0; i < batchSize; i++) {
                  double gradient = 0.0;
                  for(int j = 0; j < parameter; j++) {
                        for(int k = 0; k < parameter; k++) {
                              if (j == 0) {//theta_0 计算稍微有点差别，这里到没差别，因为
batchX[i][0]的值为 1
                                    gradient += dev(theta, batchX[i], batchY[i]);
                              } else {
                                    gradient += dev(theta, batchX[i], batchY[i]) * batchX[i][j];
                              }
                        }
                        batchTheta[i][j] = theta[j] - alpha * (gradient / parameter);
                        gradient = 0.0;
                  }

            }
            return avgTheta(batchTheta);
      }

      /**
       * 参数平均化
       * @param batchTheta
       * @return
       */
      public static double[] avgTheta(double[][] batchTheta) {
            int batchSize = batchTheta.length;
            int parameter = batchTheta[0].length;
            double[] theta = new double[parameter];
            for(int k = 0; k < parameter; k++) {
                  theta[k] = 0;
                  for(int j = 0; j < batchSize; j++) {
                        theta[k] += batchTheta[j][k];
                  }
                  theta[k] = theta[k] / batchSize;
            }
            return theta;
      }

      /**
       * 随机排序的 0 到 n-1 个整数
       * @param num
```

```
     * @return  随机排序的 0 到 n-1 个整数
     */
    public static int[] random(int num) {
        int[] rand = new int[num];
        for (int i = 0; i < num; i++) {
            rand[i] = i;
        }
        int rand1, rand2,tmp;
        for (int i = 0; i < num; i++) {
            rand1 = (int)(Math.random() * num);
            rand2 = (int)(Math.random() * num);
            if (rand1 != rand2) {
                tmp = rand[rand1];
                rand[rand1] = rand[rand2];
                rand[rand2] = tmp;
            }
        }
        return rand;
    }

    /**
     * 梯度下降训练
     * @param x_data
     * @param y_data
     * @param alpha
     * @param deviation
     * @param batchSize
     * @param iterations
     * @return
     * @throws Exception
     */
    public static double[] gdTrain(double[][] x_data, double[] y_data, double alpha, double deviation, int batchSize, int iterations) throws Exception {
        int parameter = x_data[0].length;
        int size = y_data.length;
        if (size % batchSize != 0) {
            throw new Exception("样本数必须是 batchSize 的整数倍！");
        }
        double[][] batchTheta = new double[batchSize][parameter];
        //随机初始化参数
        for(int j = 0; j < parameter; j++) {
            batchTheta[0][j] = Math.random();
        }
        for(int i = 1; i < batchSize; i++) {
```

```
                batchTheta[i] = batchTheta[0];
            }
//          for(int i = 0; i < batchSize; i++) {
//              for(int j = 0; j < parameter; j++) {
//                  batchTheta[i][j] = Math.random();
//              }
//          }
            int count = 1;
            int batchCount = 1;
            int lossEqual = 0;
            double loss = 0;
            double lastLoss = 0;
            double[][] batchX = new double[batchSize][parameter];
            double[] batchY = new double[batchSize];
            double[] theta = new double[parameter];
            while (count <= iterations) {
                for(int i = 0; i < size; i = i + batchSize) {
                    for(int j = 0; j < batchSize; j++) {
                        batchX[j] = x_data[i + j];
                        batchY[j] = y_data[i + j];
                    }

                    loss = loss(batchTheta, batchX, batchY);
                    System.out.println("step = " + batchCount + "/" + count + "\tloss = " + loss);
                    if(loss < deviation) {
                        //System.out.println("训练完成！");
                        break;
                    }
                    for(int j = 0; j < batchSize; j++) {
                        theta = updateTheta(theta, batchX[j], batchY[j], alpha);
                        batchTheta[j] = theta;
                    }
                    ++batchCount;
                }
                //alpha = alpha / 2;
                ++count;
                batchCount = 1;
                if (lastLoss == loss) {
                    ++lossEqual;
                }
                lastLoss = loss;
                if(loss < deviation) {
                    System.out.println("GD 训练完成！");
                    break;
```

```
                }
                if (lossEqual > 5) {
                    System.out.println("loss 值不在改变，GD 训练完成！");
                    break;
                }
            }
            for (int i = 0; i < theta.length; i++) {
                System.out.print(theta[i] + "\t");
            }
            System.out.println();
            return theta;
        }

        /**
         * SGD 策略一训练
         * @param x_data
         * @param y_data
         * @param alpha
         * @param deviation
         * @param batchSize
         * @param iterations
         * @return
         * @throws Exception
         */
        public static double[] sgdTrain(double[][] x_data, double[] y_data, double alpha, double
deviation, int batchSize, int iterations) throws Exception {
            int parameter = x_data[0].length;
            int size = y_data.length;
            if (size % batchSize != 0) {
                throw new Exception("样本数必须是 batchSize 的整数倍！");
            }
            double[][] batchTheta = new double[batchSize][parameter];
            //随机初始化参数
            for(int j = 0; j < parameter; j++) {
                batchTheta[0][j] = Math.random();
            }
            for(int i = 1; i < batchSize; i++) {
                batchTheta[i] = batchTheta[0];
            }
            int count = 1;
            int batchCount = 1;
            int lossEqual = 0;
            double loss = 0;
            double lastLoss = 0;
```

```
double[][] batchX = new double[batchSize][parameter];
double[] batchY = new double[batchSize];
double[] theta = new double[parameter];
int[] randoms = new int[size];
while (count <= iterations) {
    randoms = random(size);
    for(int i = 0; i < size; i = i + batchSize) {
        for(int j = 0; j < batchSize; j++) {
            batchX[j] = x_data[randoms[i + j]];
            batchY[j] = y_data[randoms[i + j]];
        }

        loss = loss(batchTheta, batchX, batchY);
        System.out.println("step = " + batchCount + "/" + count + "\tloss = " + loss);
        if(loss < deviation) {
            //System.out.println("训练完成！");
            break;
        }
        for(int j = 0; j < batchSize; j++) {
            theta = updateTheta(theta, batchX[j], batchY[j], alpha);
            batchTheta[j] = theta;
        }
        ++batchCount;
    }
    //alpha = alpha / 2;
    ++count;
    batchCount = 1;
    if (lastLoss == loss) {
        ++lossEqual;
    }
    lastLoss = loss;
    if(loss < deviation) {
        System.out.println("SGD 训练完成！");
        break;
    }
    if (lossEqual > 5) {
        System.out.println("loss 值不在改变，SGD 训练完成！");
        break;
    }
}
for (int i = 0; i < theta.length; i++) {
    System.out.print(theta[i] + "\t");
}
System.out.println();
```

```
        return theta;
    }

    /**
     * SGD 策略二训练
     * @param x_data
     * @param y_data
     * @param alpha
     * @param deviation
     * @param batchSize
     * @param iterations
     * @return
     * @throws Exception
     */
    public static double[] sgdTrain2(double[][] x_data, double[] y_data, double alpha, double deviation, int batchSize, int iterations) throws Exception {
        int parameter = x_data[0].length;
        int size = y_data.length;
        if (size % batchSize != 0) {
            throw new Exception("样本数必须是 batchSize 的整数倍！");
        }
        double[][] batchTheta = new double[batchSize][parameter];
        //随机初始化参数
        for(int j = 0; j < parameter; j++) {
            batchTheta[0][j] = Math.random();
        }
        for(int i = 1; i < batchSize; i++) {
            batchTheta[i] = batchTheta[0];
        }
        int count = 1;
        int lossEqual = 0;
        double loss = 0;
        double lastLoss = 0;
        double[][] batchX = new double[batchSize][parameter];
        double[] batchY = new double[batchSize];
        double[] theta = new double[parameter];
        int[] randoms = new int[size];
        while (count <= iterations) {
            randoms = random(size);
            for(int j = 0; j < batchSize; j++) {
                batchX[j] = x_data[randoms[j]];
                batchY[j] = y_data[randoms[j]];
            }
```

```
                loss = loss(batchTheta, batchX, batchY);
                System.out.println("step = " + count + "\tloss = " + loss);
                if(loss < deviation) {
                    System.out.println("SGD 训练完成！ ");
                    break;
                }
                for(int j = 0; j < batchSize; j++) {
                    theta = updateTheta(theta, batchX[j], batchY[j], alpha);
                    batchTheta[j] = theta;
                }
                alpha = alpha / 2;
                ++count;
                if (lastLoss == loss) {
                    ++lossEqual;
                }
                lastLoss = loss;

                if (lossEqual > 5) {
                    System.out.println("loss 值不在改变，SGD 训练完成！ ");
                    break;
                }
            }
            for (int i = 0; i < theta.length; i++) {
                System.out.print(theta[i] + "\t");
            }
            System.out.println();
            return theta;
        }

        /**
         * ASGD 策略一训练
         * @param x_data
         * @param y_data
         * @param alpha
         * @param deviation
         * @param batchSize
         * @param iterations
         * @return
         * @throws Exception
         */
        public static double[] asgdTrain(double[][] x_data, double[] y_data, double alpha, double
deviation, int batchSize, int iterations) throws Exception {
            int parameter = x_data[0].length;
            int size = y_data.length;
```

```
        if (size % batchSize != 0) {
            throw new Exception("样本数必须是 batchSize 的整数倍！");
        }
        double[] theta = new double[parameter];
        //随机初始化参数
        for(int j = 0; j < parameter; j++) {
            theta[j] = Math.random();
        }
        int count = 1;
        int batchCount = 1;
        int lossEqual = 0;
        double loss = 0;
        double lastLoss = 0;
        double[][] batchX = new double[batchSize][parameter];
        double[] batchY = new double[batchSize];
        int[] randoms = new int[size];
        while (count <= iterations) {
            randoms = random(size);
            for(int i = 0; i < size; i = i + batchSize) {
                for(int j = 0; j < batchSize; j++) {
                    batchX[j] = x_data[randoms[i + j]];
                    batchY[j] = y_data[randoms[i + j]];
                }

                loss = loss(theta, batchX, batchY);
                System.out.println("step = " + batchCount + "/" + count + "\tloss = " + loss);
                if(loss < deviation) {
                    //System.out.println("训练完成！");
                    break;
                }
                theta = updateTheta(theta, batchX, batchY, alpha);
                ++batchCount;
            }
            ++count;
            batchCount = 1;
            if (lastLoss == loss) {
                ++lossEqual;
            }
            lastLoss = loss;
            if(loss < deviation) {
                System.out.println("ASGD 训练完成！");
                break;
            }
            if (lossEqual > 5) {
```

```
                    System.out.println("loss 值不在改变，ASGD 训练完成！");
                    break;
                }
            }
            for (int i = 0; i < theta.length; i++) {
                System.out.print(theta[i] + "\t");
            }
            System.out.println();
            return theta;
        }

        /**
         * ASGD 策略二训练
         * @param x_data
         * @param y_data
         * @param alpha
         * @param deviation
         * @param batchSize
         * @param iterations
         * @return
         * @throws Exception
         */
        public static double[] asgdTrain2(double[][] x_data, double[] y_data, double alpha, double
deviation, int batchSize, int iterations) throws Exception {
            int parameter = x_data[0].length;
            int size = y_data.length;
            if (size % batchSize != 0) {
                throw new Exception("样本数必须是 batchSize 的整数倍！");
            }
            double[] theta = new double[parameter];
            //随机初始化参数
            for(int j = 0; j < parameter; j++) {
                theta[j] = Math.random();
            }

            int count = 1;
            int lossEqual = 0;
            double loss = 0;
            double lastLoss = 0;
            double[][] batchX = new double[batchSize][parameter];
            double[] batchY = new double[batchSize];
            int[] randoms = new int[size];
            while (count <= iterations) {
                randoms = random(size);
```

```
            for(int j = 0; j < batchSize; j++) {
                batchX[j] = x_data[randoms[j]];
                batchY[j] = y_data[randoms[j]];
            }

            loss = loss(theta, batchX, batchY);
            System.out.println("step = " + count + "\tloss = " + loss);
            if(loss < deviation) {
                System.out.println("ASGD 训练完成！ ");
                break;
            }

            theta = updateTheta(theta, batchX, batchY, alpha);

            ++count;
            if (lastLoss == loss) {
                ++lossEqual;
            }
            lastLoss = loss;

            if (lossEqual > 5) {
                System.out.println("loss 值不在改变，ASGD 训练完成！");
                break;
            }
        }
        for (int i = 0; i < theta.length; i++) {
            System.out.print(theta[i] + "\t");
        }
        System.out.println();
        return theta;
    }



    public static void main(String[] args) throws Exception {

        // 生成训练数据   y = a + b * x1 + c * x2
        int dataSize = 100;
        double[][] x_data = new double[dataSize][3];
        double[] y_data = new double[dataSize];
        double x1 = 0.0;
        double x2 = 0.0;
        for (int i = 0; i < dataSize; i++) {
```

```
                x_data[i][0] = 1.0;
                x1 = Math.random();
                x2 = Math.random();
                x_data[i][1] = x1;
                x_data[i][2] = x2;
                y_data[i] = 0.2 + 0.3 * x1 + 0.5 * x2;

            }

            // 生成训练数据 y = a + b * x1
//          int dataSize = 100;
//          double[][] x_data = new double[dataSize][2];
//          double[] y_data = new double[dataSize];
//          double x1 = 0.0;
//          for (int i = 0; i < dataSize; i++) {
//              x_data[i][0] = 1.0;
//              x1 = Math.random();
//              x_data[i][1] = x1;
//              y_data[i] = 0.2 + 0.3 * x1;
//
//          }

            //训练速率
            double alpha = 0.5;
            double deviation = 0.000001;
            //每次更新参数使用样本大小
            int batchSize = 20;
            //迭代次数上限
            int iterations = 10000;

            //梯度下降法，如果 batchSize 的大小等于样本总数，则为批量梯度下降法
            //gdTrain(x_data, y_data, alpha, deviation, batchSize, iterations);

            //随机梯度下降法
            //随机梯度下降法中，batchSize 不应该等于样本总数，如果等，将退化为批量梯度下降
            /**
             * 随机策略一
             * 每一轮迭代样本前，生成 0-（n-1）（n 为样本总数）的 n 个数组成的数组，并将其随机打乱，
             * 在根据打乱的数组按 batchSize 的大小取样本进行训练；
             * 该方法中，每一轮中的每一次更新参数的样本是随机的，但每一轮中一个样本只使用一次，即每一轮中，每一次更新参数的样本不会重复使用，
             * 也就是说没一轮下来将用完所有的样本
```

```
            * 但有可能出现一轮未完成就已经收敛
            *
            * 如果 batchSize = 1 为 SGD
            * 如果 batchSize > 1 为 MSGD
            */
        //sgdTrain(x_data, y_data, alpha, deviation, batchSize, iterations);

        /**
         * 随机策略二
         * 每次更新参数是从样本 n 中随机选取 batchSize 个样本进行训练
         * 当前次和下一次更新参数的样本可能存在同一个或几个样本，
         * 迭代次数少时，有的样本可能未被使用，
         * 当迭代次数足够多时，应该所有的样本都已使用到
         *
         * 如果 batchSize = 1 为 SGD
         * 如果 batchSize > 1 为 MSGD
         */
        //sgdTrain2(x_data, y_data, alpha, deviation, batchSize, iterations);

        //平均随机梯度下降
        /**
         *  随机策略一
         */
        asgdTrain(x_data, y_data, alpha, deviation, batchSize, iterations);

        /**
         *  随机策略二
         */
        asgdTrain2(x_data, y_data, alpha, deviation, batchSize, iterations);



    }

}
```