《数据仓库与数据挖掘》实验报告

姓名：	朱志儒	学号：	SA20225085	日期：	2020/11/30
上机题目：	基于梯度下降和随机梯度下降的线性回归算法实现				

操作环境：
OS：Window 10
CPU：AMD Ryzen 5 3600X 6-Core Processor 4.25GHz
GPU：GeForce RTX 2070 super

一、基础知识：

1. 线性回归：

假设多元线性回归中有多个自变量(x_0, x_1, x_2, …, x_n)，那么多元线性回归模型的假设函数可以写成：

$$h(x)=\beta_0+\beta_1x_1+\beta_2x_2+\ldots+\beta_nx_n$$

损失函数：

$$J(\beta_0,\beta_1)=\frac{1}{2m}\sum_{i=1}^{m}[h(x^{(i)})-y^{(i)}]^2$$

2. 梯度下降：

n 元实值函数 g(x)在 n 维空间中变化速度最快的方向：

$$\nabla_x g \triangleq \left(\frac{\partial g}{\partial x_1},\frac{\partial g}{\partial x_2},\ldots,\frac{\partial g}{\partial x_n}\right)^t$$

第一步，初始化，设置算法相关参数/超参数，主要包括两个参数：算法停止准则 T，当执行 T 次循环时，算法终止；下降步长，该序列一般为递减。

第二步，初始化，初态 X_0，初态 X_0 可设置为某个特殊值，也可以用随机值。

第三步，循环迭代：

for s = 1, 2, 3, …, T

$X_{s+1}=X_s-\lambda_s\nabla_X g(X_s)$，其中，

$$\nabla_X g=\left(\frac{\partial g}{\partial x_1},\frac{\partial g}{\partial x_2},\ldots,\frac{\partial g}{\partial x_n}\right)^t \text{ and } X=(x_1,x_2,\ldots,x_n)^t$$

end for

3. 随机梯度下降：

计算梯度方向时，只用了一个随机样本的信息，替代经典梯度下降算法要计算所有训练数据集对梯度方向的贡献。

即：

$$W_{s+1}=W_s-\lambda_s\frac{\sum_{(x,y)\in D_{train}}2(W_s\cdot\phi(x)-y)\phi(x)}{|D_{train}|}\Longrightarrow W_{s+1}=$$

$$W_s-\lambda_s\nabla_W((W_s\cdot\phi(x_i)-y_i)^2)$$

时间性能得到提升，不需要每次循环都载入一个训练数据集，付出的代价是更多的循环次数。

二、实验过程：

实验采用的数据集：

```
V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	target
0.566	0.016	-0.143	0.407	0.452	-0.901	-1.812	-2.36	-0.436	-2.114	-0.94	-0.307	-0.073	0.55	-0.484	0.0	-1.707	-1.162	-0.573	-0.991	0.61
0.968	0.437	0.066	0.566	0.194	-0.893	-1.566	-2.36	0.332	-2.114	0.188	-0.455	-0.134	1.109	-0.488	0.0	-0.977	-1.162	-0.571	-0.836	0.588
1.013	0.568	0.235	0.37	0.112	-0.797	-1.367	-2.36	0.396	-2.114	0.874	-0.051	-0.072	0.767	-0.493	-0.212	-0.618	-0.897	-0.564	-0.558	0.5
0.733	0.368	0.283	0.165	0.599	-0.679	-1.2	-2.086	0.403	-2.114	0.011	0.102	-0.014	0.769	-0.371	-0.162	-0.429	-0.897	-0.574	-0.564	0.2
0.684	0.638	0.26	0.209	0.337	-0.454	-1.073	-2.086	0.314	-2.114	-0.251	0.57	0.199	-0.349	-0.342	-0.138	-0.391	-0.897	-0.572	-0.394	0.1
0.445	0.627	0.408	0.22	0.458	-1.056	-1.009	-1.896	0.481	-2.114	-0.511	-0.564	0.294	0.912	-0.345	0.111	-0.333	-1.029	-0.573	-0.516	0.0
0.889	0.416	0.64	0.356	0.224	-0.893	-0.812	-1.823	0.729	-2.114	-0.256	-0.278	0.425	0.632	-0.3	0.111	-0.333	-1.428	-0.586	-0.544	0.1
0.984	0.529	0.704	0.438	0.258	-0.917	-0.682	-1.721	0.753	-2.114	-0.067	-0.24	0.272	0.78	-0.387	0.244	0.065	-1.162	-0.579	-0.465	0.2
0.948	0.85	0.584	0.459	0.591	-0.523	-0.591	-1.524	0.763	-2.114	0.205	0.422	0.387	-0.288	-0.264	0.293	0.166	-1.162	-0.566	-0.173	0.2
1.157	1.055	0.638	0.617	1.483	-0.731	-0.612	-1.524	0.968	-2.114	0.145	0.179	0.688	-0.14	-0.289	0.317	0.195	-0.897	-0.567	-0.557	0.2
1.116	1.112	0.612	0.639	0.919	-0.895	-0.656	-1.418	0.891	-2.114	0.122	-0.275	1.083	0.429	-0.256	1.017	0.161	-0.963	-0.575	-0.381	0.2
1.093	1.12	0.522	0.797	0.953	-0.619	-0.591	-1.39	1.044	-2.114	0.817	0.431	1.472	-0.45	-0.332	1.112	0.282	-1.118	-0.568	0.51	0.1
-0.632	-0.959	-0.474	1.194	-0.17	-0.572	-0.779	-1.35	-1.034	-2.114	0.025	0.514	-0.419	-0.099	-0.371	1.592	0.293	-0.897	-0.583	0.466	-0.
-0.999	-1.343	-0.454	1.156	-0.523	-0.786	-0.988	-1.463	-1.948	-2.114	-1.603	0.036	-0.751	0.569	-0.402	2.064	-0.929	-0.897	0.064	0.423	-0.
-0.234	-0.248	-0.271	0.94	0.174	-0.61	-1.102	-1.463	-0.987	-2.114	-1.004	0.419	0.34	0.436	-0.339	2.064	-1.075	-0.897	-1.304	0.396	-0.
-0.898	-1.057	-0.655	0.94	-0.544	-0.654	-1.245	-1.634	-1.784	-2.114	-0.25	0.275	-0.247	0.545	-0.258	1.98	-1.261	-0.897	-1.296	0.108	-0.
-0.403	-0.486	-0.304	0.951	-0.294	-0.488	-1.29	-1.686	-1.105	-2.114	-0.732	0.688	0.357	-0.288	-0.245	1.626	-1.463	-0.897	-1.31	0.6	-0.483
-1.158	-1.538	-0.585	0.797	-0.818	-0.668	-1.416	-1.759	-2.33	-2.114	-0.917	0.248	-0.591	0.59	-0.264	1.626	-1.707	-1.162	-1.306	0.413	-0.
-0.638	-0.312	-0.059	0.831	-0.223	-0.907	-1.403	-1.878	-1.317	-2.114	-1.014	-0.306	0.189	1.305	-0.29	0.721	-1.66	-1.162	-1.273	0.275	-0.
0.375	0.422	0.369	0.929	0.132	-1.599	-1.263	-1.878	-0.323	-2.114	-1.0	-1.372	0.528	1.123	-0.265	0.665	-1.268	-1.162	-0.389	0.197	0.0
0.211	0.385	0.491	1.182	-0.035	-1.287	-1.127	-1.84	-0.612	-2.114	-0.11	-0.868	0.453	1.147	-0.218	0.055	-1.077	-1.251	-0.388	0.202	0.0
-0.041	0.093	0.354	1.264	-0.221	-1.078	-1.028	-1.801	-0.749	-2.114	0.205	-0.544	0.201	1.428	-0.294	-0.251	-0.887	-1.428	2.684	0.256	-0.
0.369	0.479	0.723	1.36	0.072	-0.747	-0.888	-1.746	0.099	-2.114	-0.114	0.071	0.072	1.223	-0.261	-0.251	-0.653	-1.162	3.619	-0.048	-0.
-0.143	-0.345	0.363	1.313	0.04	-0.516	-0.754	-1.617	-0.643	-2.114	-0.165	0.546	-0.035	1.009	-0.331	-0.433	-0.527	-1.162	4.072	-0.203	-0.
-2.028	-2.665	-0.467	0.435	-1.479	-0.562	-0.798	-1.617	-2.754	-2.114	1.524	0.343	-1.657	1.033	-0.206	-0.318	-0.812	0.165	0.053	0.0	-1.516
-1.692	-2.18	-0.595	-0.669	-0.854	-0.574	-0.899	-1.61	-2.546	-2.114	-2.108	0.196	-0.82	0.585	-0.335	0.026	-1.357	-0.897	0.568	0.324	-1.
0.618	0.245	0.382	-0.131	0.966	-1.081	-0.892	-1.64	0.291	-2.114	-2.577	-0.488	0.205	0.56	-0.279	0.595	-1.375	-1.029	0.524	0.282	0.0
1.196	1.166	0.748	-0.596	1.421	-1.184	-0.808	-1.682	0.587	-2.114	-1.64	-1.198	0.928	2.293	-0.14	0.595	-0.741	-1.162	0.64	0.555	0.2
1.106	1.177	1.033	-0.045	1.124	-1.814	-0.49	-1.549	0.544	-2.114	0.654	-2.275	1.454	1.477	-0.15	1.418	-0.09	-1.162	0.066	-0.336	0.1
0.462	0.48	0.593	-0.319	0.333	-1.293	-0.382	-1.549	-0.16	-2.114	1.699	-1.265	0.695	1.479	-0.255	1.368	-0.068	0.165	0.069	0.055	-0.
0.335	0.253	0.61	-0.518	0.159	-1.374	-0.297	-1.335	-0.255	-2.114	0.705	-1.035	0.687	0.98	-0.356	1.22	-0.127	-0.366	1.974	-0.073	-0.
-0.254	-0.148	0.237	-0.782	0.032	-0.591	-0.324	-1.305	-0.503	-2.114	0.85	0.439	0.44	0.806	-0.3	0.87	-0.41	-0.897	1.056	-0.299	-1.
-0.351	-0.485	0.302	-1.446	-0.031	-1.519	-0.438	-1.262	-0.812	-2.114	-1.41	-1.29	-0.096	0.771	-0.297	0.87	-0.752	-1.162	0.985	-0.051	-1.
-0.261	-0.443	-0.049	-1.513	0.013	-0.857	-0.535	-1.299	-0.997	-2.114	-0.726	-0.159	0.027	0.635	-0.291	0.495	-0.809	-1.162	-1.13	0.234	-1.
0.369	0.615	0.706	-1.63	0.607	-1.296	-0.535	-1.299	0.176	-2.114	-0.896	-0.963	0.485	0.868	-0.284	0.448	-0.675	-1.428	0.532	-0.52	-1.
0.822	0.804	1.5	-1.6	0.631	-1.757	-0.37	-1.32	0.578	-2.114	0.244	-1.963	0.786	1.243	-0.23	0.309	-0.675	-1.428	0.532	-0.693	-1.074
0.776	0.855	1.028	-1.634	0.458	-0.682	-0.194	-1.241	0.618	-2.114	1.318	0.232	0.362	0.646	-0.226	0.114	0.189	-1.428	0.529	-0.274	-0.
0.494	0.729	0.786	-1.754	0.341	-0.72	-0.189	-1.13	0.149	-2.114	0.494	0.084	0.395	1.365	-0.116	0.114	-0.031	-0.897	0.525	-0.54	-1.
0.386	0.329	0.658	-1.942	0.261	-0.515	-0.193	-1.073	0.063	-2.114	0.176	0.453	0.571	1.22	-0.15	-0.009	-0.111	-0.897	0.535	-0.32	-1.
-0.648	-0.522	0.173	-1.925	-0.397	-0.743	-0.309	-1.073	-0.994	-2.114	-0.533	-0.097	0.167	1.612	-0.241	-0.066	-0.467	-1.428	2.226	-0.347	-1.
0.085	0.073	1.313	-1.867	-0.405	-0.928	-0.269	-1.099	0.165	-2.114	-1.127	-0.151	0.005	1.538	-0.21	-0.239	-0.57	-1.693	0.077	-0.572	-1.
-0.436	-0.25	0.506	-1.813	-0.374	-0.567	-0.254	-1.076	-0.82	-2.114	0.179	0.312	-0.08	1.099	-0.298	-0.689	-0.121	-1.693	-3.512	0.406	-1.
-0.014	0.099	0.854	-1.715	-0.171	-0.914	-0.285	-1.045	-0.439	-2.114	-0.187	-0.375	0.448	1.314	-0.16	-0.689	-0.483	-1.428	0.072	-0.104	-0.
0.507	0.634	1.096	-1.602	0.191	-1.579	-0.227	-1.009	0.348	-1.036	-0.642	-1.648	0.326	1.297	-0.172	-0.627	-0.244	-1.428	0.142	0.717	-0.
0.669	0.675	0.82	-1.324	0.323	-1.941	-0.099	-1.009	0.112	-1.036	-0.237	-2.413	0.604	1.16	-0.248	-0.246	-0.076	-1.428	1.437	0.539	0.3
0.809	0.642	1.002	-1.214	0.332	-1.776	-0.02	-0.916	0.071	-0.821	0.073	-2.073	0.688	2.058	-0.195	-0.246	-0.098	-1.428	1.442	0.397	0.3
0.827	0.602	1.66	-1.131	0.114	-1.474	0.221	-0.864	0.516	-0.821	0.687	-1.362	0.561	1.154	-0.346	0.064	-0.009	-1.428	1.439	0.511	0.3
```

本实验使用 C++语言，采用二维数组存储数据集，采用长度为 39 的一维数组存储 w，其中，w0~w37 分别对应输入数据中的 v0~v37 的系数 w_i，w38 表示的是偏移量 b。规定迭代次数为 1000 次，采用梯度下降算法更新数组 w，每次迭代后打印模型的 loss。

三、结果分析：

基于梯度下降的线性回归算法：

```
迭代次数：977   loss: 0.0605286
迭代次数：978   loss: 0.06052
迭代次数：979   loss: 0.0605115
迭代次数：980   loss: 0.060503
迭代次数：981   loss: 0.0604945
迭代次数：982   loss: 0.0604861
迭代次数：983   loss: 0.0604776
迭代次数：984   loss: 0.0604692
迭代次数：985   loss: 0.0604608
迭代次数：986   loss: 0.0604525
迭代次数：987   loss: 0.0604441
迭代次数：988   loss: 0.0604358
迭代次数：989   loss: 0.0604275
迭代次数：990   loss: 0.0604192
迭代次数：991   loss: 0.060411
迭代次数：992   loss: 0.0604028
迭代次数：993   loss: 0.0603946
迭代次数：994   loss: 0.0603864
迭代次数：995   loss: 0.0603782
迭代次数：996   loss: 0.0603701
迭代次数：997   loss: 0.060362
迭代次数：998   loss: 0.0603539
迭代次数：999   loss: 0.0603458
w 0:     0.369548   w 1:     0.197901   w 2:     0.339583   w 3:     0.155296   w 4:  -0.00953977   w 5:     0.351832
w 6:     0.102488   w 7:     -0.11807   w 8:    0.0120913   w 9:    0.0631521   w10:     0.156489   w11:    -0.147061
w12:     0.165503   w13:    0.0484207   w14:    0.0488732   w15:    0.0181929   w16:    -0.145218   w17:    0.0498367
w18:   0.00229798   w19:   -0.0204678   w20:   -0.0310788   w21:    0.0211816   w22:    0.0286429   w23:  -0.00168216
w24:   -0.0484314   w25:    0.0335265   w26:   -0.0152527   w27:     0.593635   w28:   0.00150232   w29:   -0.0733345
w30:    0.0191366   w31:   -0.0595272   w32:  -0.00696867   w33:    0.0207403   w34:  0.000801273   w35:  -0.00517766
w36:   -0.0643518   w37:    0.0287131   w38:   -0.0251195
```

基于随机梯度下降的线性回归算法：

```
迭代次数：978  loss：50.2992
迭代次数：979  loss：50.3
迭代次数：980  loss：50.3007
迭代次数：981  loss：50.3015
迭代次数：982  loss：50.3023
迭代次数：983  loss：50.303
迭代次数：984  loss：50.3038
迭代次数：985  loss：50.3045
迭代次数：986  loss：50.3053
迭代次数：987  loss：50.306
迭代次数：988  loss：50.3068
迭代次数：989  loss：50.3076
迭代次数：990  loss：50.3083
迭代次数：991  loss：50.3091
迭代次数：992  loss：50.3098
迭代次数：993  loss：50.3106
迭代次数：994  loss：50.3113
迭代次数：995  loss：50.3121
迭代次数：996  loss：50.3129
迭代次数：997  loss：50.3136
迭代次数：998  loss：50.3144
迭代次数：999  loss：50.3151
w 0:   0.98042   w 1:  0.985419  w 2:  0.977222  w 3:   1.02567  w 4:  0.980732  w 5:   1.00635
w 6:  0.962604   w 7:  0.957649  w 8:  0.972511  w 9:   1.00071  w10:  0.935954  w11:   1.02135
w12:  0.975614   w13:  0.966191  w14:  0.999647  w15:  0.983581  w16:  0.962652  w17:   1.00578
w18:  0.984356   w19:   1.04741  w20:   1.02128  w21:   1.03186  w22:   1.00426  w23:   1.00612
w24:  0.998742   w25:   1.00526  w26:   1.01048  w27:  0.996817  w28:  0.970414  w29:  0.980497
w30:  0.968783   w31:   0.96746  w32:   1.00558  w33:   1.00708  w34:   1.00271  w35:   1.00615
w36:  0.934231   w37:   1.01422  w38:   1.01691
```

从上述的两个方法的结果可以知道，基于梯度下降的线性回归算法的 loss 比基于随机梯度下降的线性回归算法的要小，也就是说前者的拟合效果更好。这是因为随机梯度下降每次更新 w 数组时只使用一个随机样本，所以找到局部最优解的效率较低。本次实验规定迭代次数为 1000，可能还没找到局部最优解训练就结束了，所以若要达到与梯度下降算法同样的效果，就需要增加迭代次数。

附录

算法源代码（C/C++/JAVA 描述）：

基于梯度下降的线性回归算法：

```
double predict(double* x, double* w, int m) {
    double y = w[m];
    for (int i = 0; i < m; ++i)
        y += x[i] * w[i];
    return y;
}

void gradient_descent(double** x, double* y, double* w, double lambda, int n, int m) {
    double* difference = new double[n];
    for (int i = 0; i < n; ++i)
        difference[i] = predict(x[i], w, m) - y[i];
    double* errors = new double[m + 1];
    for (int i = 0; i < m + 1; ++i)
        errors[i] = 0;
    for (int i = 0; i < n; ++i)      //w[m]
        errors[m] += difference[i];
    for (int i = 0; i < m; ++i)  //w[0]~w[m-1]
        for (int j = 0; j < n; ++j)
            errors[i] += difference[j] * x[j][i];
```

```cpp
    for (int i = 0; i < m + 1; ++i)
        w[i] -= lambda * 2.0 / n * errors[i];
    delete[] difference;
    delete[] errors;
}

void train(double** x, double* y, double* w, double lambda, int n, int m, int iter) {
    for (int i = 0; i < iter; ++i) {
        gradient_descent(x, y, w, lambda, n, m);
        double loss = 0;
        for (int i = 0; i < n; ++i)
            loss += (predict(x[i], w, m) - y[i]) * (predict(x[i], w, m) - y[i]);
        cout << "迭代次数：" << i << "  " << "loss: " << loss << endl;
    }
}

void save_w(double* w, int m) {
    ofstream outfile("w1.txt");
    for (int i = 0; i <= m; ++i)
        outfile << w[i] << ' ';
}

void init_w(double* w, int m) {
    ifstream infile("w1.txt");
    if (!infile)
        for (int i = 0; i <= m; ++i)
            w[i] = 1;
    else
        for (int i = 0; i <= m; ++i)
            infile >> w[i];
}

int main() {
    int n = 2888, m = 38;
    double** x_array = new double*[n];
    for (int i = 0; i < n; ++i)
        x_array[i] = new double[m];
    double* y_array = new double[n];
    ifstream infile("data_train.txt");
    string head[39];
    for (int i = 0; i < m + 1; ++i)
```

```
        infile >> head[i];
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j)
            infile >> x_array[i][j];
        infile >> y_array[i];
    }
    double* w = new double[m + 1];
    init_w(w, m);
    double lambda = 0.01;
    int iter = 10000;
    train(x_array, y_array, w, lambda, n, m, iter);
    save_w(w, m);
    for (int i = 0; i <= m; ++i) {
        cout << "w" << setw(2) << i << ": " << setw(12) << w[i] << "  ";
        if (i % 6 == 5)
            cout << endl;
    }
    return 0;
}
```

基于随机梯度下降线性回归算法：

```
double predict(double* x, double* w, int m) {
    double y = w[m];
    for (int i = 0; i < m; ++i)
        y += x[i] * w[i];
    return y;
}

void random_gradient_descent(double** x, double* y, double* w, double lambda, int n, int m) {
    srand(time(NULL));
    int index = rand() % (n + 1);
    double difference = predict(x[index], w, m) - y[index];

    double* errors = new double[m + 1];
    for (int i = 0; i < m + 1; ++i)
        errors[i] = 0;
    errors[m] = difference;
    for (int i = 0; i < m; ++i)
        errors[i] += difference * x[index][i];
    for (int i = 0; i < m + 1; ++i)
        w[i] -= lambda * 2.0 / n * errors[i];
```

```
    delete[] errors;
}

void train(double** x, double* y, double* w, double lambda, int n, int m, int iter) {
    for (int i = 0; i < iter; ++i) {
        random_gradient_descent(x, y, w, lambda, n, m);
        double loss = 0;
        for (int i = 0; i < n; ++i)
            loss += (predict(x[i], w, m) - y[i]) * (predict(x[i], w, m) - y[i]);
        cout << "loss: " << loss << endl;
    }
}

void save_w(double* w, int m) {
    ofstream outfile("w2.txt");
    for (int i = 0; i <= m; ++i)
        outfile << w[i] << ' ';
}

void init_w(double* w, int m) {
    ifstream infile("w2.txt");
    if (!infile)
        for (int i = 0; i <= m; ++i)
            w[i] = 1;
    else
        for (int i = 0; i <= m; ++i)
            infile >> w[i];
}

int main() {
    int n = 2888, m = 38;
    double** x_array = new double* [n];
    for (int i = 0; i < n; ++i)
        x_array[i] = new double[m];
    double* y_array = new double[n];
    ifstream infile("data_train.txt");
    string head[39];
    for (int i = 0; i < m + 1; ++i)
        infile >> head[i];
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j)
            infile >> x_array[i][j];
```

```
        infile >> y_array[i];
    }
    double* w = new double[m + 1];
    init_w(w, m);
    double lambda = 0.01;
    int iter = 10000;
    train(x_array, y_array, w, lambda, n, m, iter);
    save_w(w, m);
    for (int i = 0; i <= m; ++i) {
        cout << "w" << setw(2) << i << ": " << setw(12) << w[i] << "  ";
        if (i % 6 == 5)
            cout << endl;
    }
    return 0;
}
```