《数据仓库与数据挖掘》实验报告

姓名：	朱志儒	学号：	SA20225085	日期：	2020/12/7
上机题目：	线性分类算法实现				

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($x_0, x_1, x_2, \ldots, x_n$)，那么多元线性回归模型的假设函数可以写成：

$$h(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

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：

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

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，

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

end for

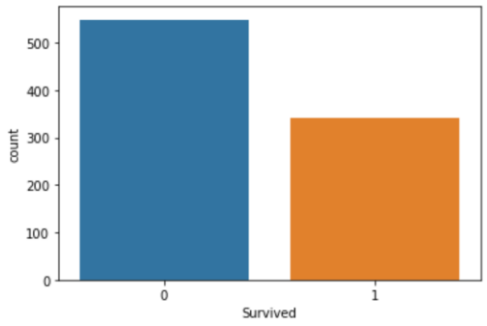
二、实验过程：

观察训练集数据集的分布情况：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

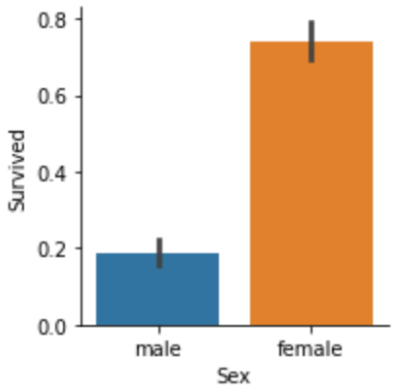
从图中可以看出 Age、Cabin、Embarked 列的数据中出现空缺。

观察训练集中 Survived 列的分布情况：



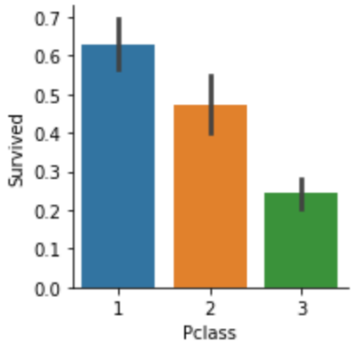
从图中可看出值为 0 的数量比值为 1 的要多。

观察训练集中 Sex 列与 Survived 列之间的关系：



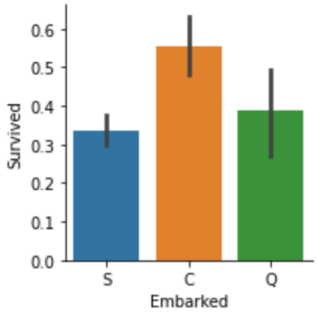
从图中可以看出女性的存活率比男性要高出很多。

观察训练集中 Pclass 列与 Survived 列之间的关系：



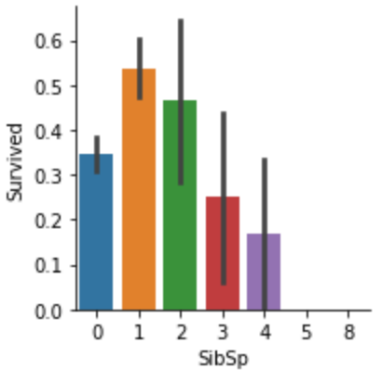
从图中可以看出 Pclass 值为 1 的存活率最高，值为 2 的存活率次之，值为 3 的存活率最低。

观察训练集中 Embarked 列与 Survived 列之间的关系：



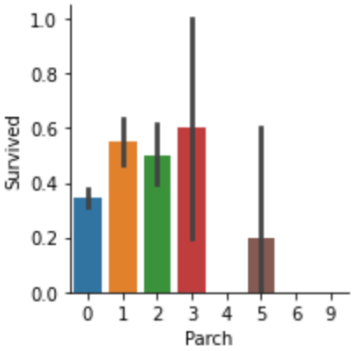
从图中可以看出 Embarked 值为 C 的存活率最高，值为 Q 的存活率次之，值为 S 的存活率最低。

观察训练集中 SibSp 列与 Survived 列之间的关系：



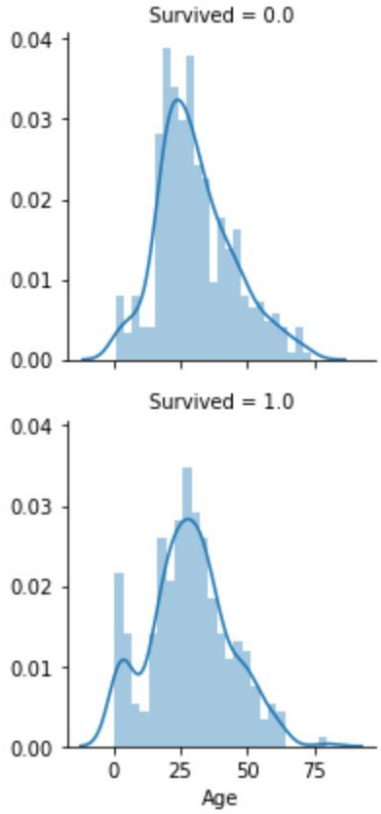
从图中可以看出 SibSp 值为 1 的存活率最高，值为 2 的存活率次之，之后依次是 0，3，4，5，8。

观察训练集中 Parch 列与 Survived 列之间的关系：



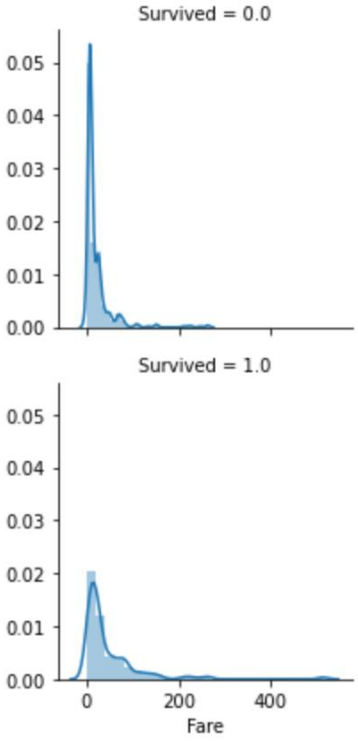
从图中可以看出 Parch 值为 3 的存活率最高，值为 1 的次之，之后依次是 2，0，5，4，6，9。

观察训练集中 Age 列与 Survived 列之间的关系：



从图中可以看出年龄在 0~15 岁的存活率较高、死亡率较低，年龄在 15~30 岁的存活率和死亡率都比较高，年龄在 60~75 岁的存活率较高、死亡率较低。

观察训练集中 Fare 列与 Survived 列之间的关系：



从图中可以看出 Fare 值越高生存率也越高。

将训练集和测试集的数据整合起来以填充其中的空缺值，然后将 Sex 列中的 male、female 分别替换为 0、1，将 Embarked 列中的 S、C、Q 分别替换为 0、1、2。

查看整个数据集中数据的缺失情况：

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

显然 Cabin 有 1014 个缺失值，Age 有 263 个缺失值，Fare 有 1 个缺失值，Embarked 有 2 个缺失值，Survived 有 418 个缺失值（这是测试集中需要预测的部分）。

找到 Fare 中缺失值的数据情况：

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1043	1044	NaN	3	Storey, Mr. Thomas	0	60.5	0	0	3701	NaN	NaN	0.0

选择 Pclass 为 3、Embarked 为 0，Sex 为 0 的数据中 Fare 的平均值来填充该缺失值。

将数据集中 Fare 的取值分成 4 组，并计算他们与生存率的关系：

```
FareLimit
(-0.001, 7.896]      0.237389
(7.896, 14.454]      0.258567
(14.454, 31.275]     0.448171
(31.275, 512.329]    0.591331
Name: Survived, dtype: float64
```

添加一个名为 FareLimit 的列，根据 Fare 的取值确定 FareLimit 的值，即当 Fare <= 7.896 时，FareLimit = 0，当 7.896 < Fare <= 14.454 时，FareLimit = 1，当 14.454 < Fare <= 31.275 时，FareLimit = 2，当 Fare > 31.275 时，FareLimit = 3。这样就可以将 Fare 的取值范围缩放到[0，3]。

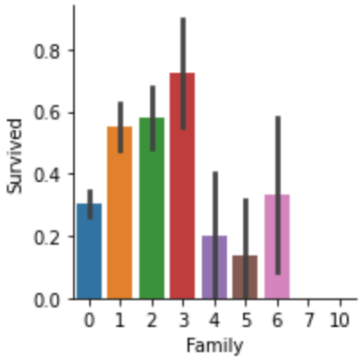
找到 Embarked 中缺失值的数据情况：

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1.0	1	Icard, Miss. Amelie	1	38.0	0	0	113572	80.0	B28	NaN
829	830	1.0	1	Stone, Mrs. George Nelson (Martha Evelyn)	1	62.0	0	0	113572	80.0	B28	NaN

选择 Pclass 为 1、Sex 为 1 的数据中 Embarked 的众数来填充该缺失值。

添加新特征 Family 表示家人数目，Family 的值是 SibSp 和 Parch 之和，观察

其与存活率的关系：



根据图中的信息可知 Family 值为 3 时存活率最高、值为 2 时次之，之后依次是 1、6、0、4、5、7、10。

添加新特征 Alone 表示是否独自一人旅行，Alone 的值由 Family 来确定，若 Family 为 0 则 Alone 为 1；若 Family 不为 0 则 Alone 为 0。

为填充 Age 中的缺失值，选择 Pclass 和 Family 取值与缺失数据相同的数据的 Age 的均值进行填充，然后将整个数据集中的 Age 分成 5 组，并计算他们与生存率的关系：

```
AgeLimit
(0.0902, 16.136]    0.534722
(16.136, 32.102]    0.339335
(32.102, 48.068]    0.401235
(48.068, 64.034]    0.452830
(64.034, 80.0]      0.076923
Name: Survived, dtype: float64
```

添加一个名为 AgeLimit 的列，根据 Age 的取值确定 AgeLimit 的值，即当 Age <= 16 时，AgeLimit = 0，当 16 < Age <= 32 时，AgeLimit = 1，当 32 < Age <= 48 时，AgeLimit = 2，当 48 < Age <= 64 时，AgeLimit = 3，当 Age > 64 时，AgeLimit = 4，这样就可以将 Age 的取值范围缩放到[0, 4]。

由于不同的乘客具有不同的 Ticket、Name、PassengerId，Cabin 缺失数据过多，他们不能为分类器提供更多的有用信息，所以删除无用的列：Age、Fare、Ticket、Cabin、Name、PassengerId。

将具有多个取值的列：Pclass、Sex、Embarked、Family、AgeLimit、FareLimit 使用 one-hot 编码以增加数据集的维度且缩小他们的取值范围到{0, 1}。

本次实验的分类器采用的是线性模型，将不同列的输入当做不同的 x_i，给予对应的不同的权重 w_i，计算它们的加权平均和 y，而正确的 y 值是 Survived 列中的 0、1 值。

损失函数采用的是平方损失函数，利用梯度下降法更新 w 的值。

三、结果分析：

迭代 10000 次以达到更好的效果：

```
loss: 0.06897951939859066
loss: 0.06897947960603303
loss: 0.06897943983361611
loss: 0.06897940008132968
loss: 0.06897936034916338
loss: 0.06897932063710711
loss: 0.06897928094515063
loss: 0.06897924127328366
loss: 0.06897920162149601
loss: 0.06897916198977752
loss: 0.06897912237811796
loss: 0.06897908278650704
loss: 0.06897904321493474
loss: 0.06897900366339077
loss: 0.06897896413186494
loss: 0.0689789246203471
loss: 0.06897888512882708
loss: 0.06897884565729474
loss: 0.06897880620573987
loss: 0.06897876677415231
loss: 0.06897872736252193
loss: 0.06897868797083863
loss: 0.06897864859909221
loss: 0.06897860924727255
loss: 0.06897856991536952
loss: 0.06897853060337301
```

显然 loss 是逐渐收敛的，将测试集数据输入至分类器结果如下：

```
测试集正确率： 0.7679425837320575
```

分类效果还是不错的。

附录

算法源代码（C/C++/JAVA 描述）：

```
1.  trainData = pd.read_csv("train.csv")
2.  testData = pd.read_csv("test.csv")
3.  # 将训练集和测试集整合
4.  data = pd.concat([trainData, testData], axis=0).reset_index
    (drop=True)
5.  # male: 0, female: 1
6.  data['Sex'].replace(['male', 'female'], [0, 1], inplace=Tru
    e)
7.  # S: 0, C: 1, Q: 2
8.  data['Embarked'].replace(['S', 'C', 'Q'], [0, 1, 2], inplac
    e=True)
9.
10. # print(data[data['Fare'].isnull()])
```

```
# Pclass: 3, Embarked: 0, Sex: 0
# 填补 Fare 为 NaN 的数据
data['Fare'] = data['Fare'].fillna(
    np.mean(data[((data['Pclass'] == 3) & (data['Embarked'] == 0) & (data['Sex'] == 0))]['Fare']))
# print(data[data['Fare'].isnull()])
# Empty DataFrame

# data['FareLimit'] = pd.qcut(data['Fare'], 6)
# print(data.groupby(['FareLimit'])['Survived'].mean())
# 使用 FareLimit 替代 Fare
data['FareLimit'] = 0
data.loc[data['Fare'] <= 8.662, 'FareLimit'] = 0
data.loc[(data['Fare'] > 8.662) & (data['Fare'] <= 14.454), 'FareLimit'] = 1
data.loc[(data['Fare'] > 14.454) & (data['Fare'] <= 53.1), 'FareLimit'] = 2
data.loc[data['Fare'] > 53.1, 'FareLimit'] = 3

# print(data[data['Embarked'].isnull()])
# Pclass: 1, Sex: 1
# 填补 Embarked 为 NaN 的数据
data['Embarked'] = data['Embarked'].fillna(
    stats.mode(data[((data['Pclass'] == 1) & (data['Sex'] == 1))]['Embarked'])[0][0])
# print(data[data['Embarked'].isnull()])
# Empty DataFrame

# 添加新特征：家人 Family
data['Family'] = data['SibSp'] + data['Parch']
data['Family'].replace([0, 1, 2, 3, 4, 5, 6, 7, 10], [0, 1, 1, 1, 0, 2, 0, 2, 2], inplace=True)

# 添加新特征：单身 Alone
data["Alone"] = [1 if i == 0 else 0 for i in data["Family"]]

# 填补 Age 为 NaN 的数据
dataAgeNanIndex = data[data['Age'].isnull()].index
for i in dataAgeNanIndex:
    # 取 Pclass、Family 相同的数据的平均值
    meanAge = data['Age'][
        (data['Pclass'] == data.iloc[i]['Pclass']) & (data['Family'] == data.iloc[i]['Family'])].mean()
```

```
48.     data['Age'].iloc[i] = meanAge
49.
50. # data['AgeLimit'] = pd.cut(data['Age'], 5)
51. # print(data.groupby(['AgeLimit'])['Survived'].mean())
52. # 使用 AgeLimit 替代 Age
53. data['AgeLimit'] = 0
54. data.loc[data['Age'] <= 16, 'AgeLimit'] = 0
55. data.loc[(data['Age'] > 16) & (data['Age'] <= 32), 'AgeLimi
    t'] = 1
56. data.loc[(data['Age'] > 32) & (data['Age'] <= 48), 'AgeLimi
    t'] = 2
57. data.loc[(data['Age'] > 48) & (data['Age'] <= 60), 'AgeLimi
    t'] = 3
58. data.loc[data['Age'] > 60, 'AgeLimit'] = 4
59.
60. # 删除无用列
61. data.drop(labels=["Age", "Fare", "Ticket", "Cabin", "Name",
     "PassengerId"], axis=1, inplace=True)
62.
63. # one-hot 编码
64. data = pd.get_dummies(data, columns=['Pclass'])
65. data = pd.get_dummies(data, columns=['Sex'])
66. data = pd.get_dummies(data, columns=['Embarked'])
67. data = pd.get_dummies(data, columns=['Family'])
68. data = pd.get_dummies(data, columns=['AgeLimit'])
69. data = pd.get_dummies(data, columns=['FareLimit'])
70.
71. trainX = data[:len(trainData)].drop(labels='Survived', axis
    =1)
72. trainY = data[:len(trainData)]['Survived']
73. testY = data[len(trainData):]['Survived']
74. testX = data[len(trainData):].drop(labels='Survived', axis=
    1)
75.
76. trainX = trainX.values
77. trainY = trainY.values
78. testX = testX.values
79. testY = testY.values.tolist()
80. n_samples = len(trainX)
81. eta = 0.1
82. iter = 10000
83.
84. trainX = np.c_[np.ones(n_samples), trainX]
85. n_features = trainX.shape[-1]
```

```
theta = np.ones(n_features)
loss_ = [0]
for i in range(iter):
    errors = trainX.dot(theta) - trainY
    loss = 1 / (2 * n_samples) * errors.dot(errors)
    delta_loss = loss - loss_[-1]
    loss_.append(loss)
    print("loss:", np.abs(loss))
    gradient = 1 / n_samples * trainX.T.dot(errors)
    theta -= eta * gradient

testX = np.c_[np.ones(len(testX)), testX]
predict = [1 if i >= 0.5 else 0 for i in testX.dot(theta).tolist()]
count = 0.0
for i in range(len(predict)):
    if testY[i] != predict[i]:
        count += 1.0
print("测试集正确率:", 1 - count / len(predict))
```