浙江大学

课程项目报告

课程名称：数据挖掘导论

姓　　名：张溢弛

学　　院：计算机科学与技术学院

专　　业：软件工程

学　　号：3180103772

指导教师：李石坚

2021 年 6 月 28 日

浙江大学 课程项目报告

目录

一 选题介绍

1.1 选题背景

我的《数据挖掘导论》课程大作业的选题是 **Titanic 生存预测竞赛** (Titanic - Machine Learning from Disaster)，这是 Kaggle 上面一个非常经典的入门级比赛，在给出一系列乘客的特征的条件下来预测乘客的存活情况，实质上是数据挖掘和机器学习中最经典的二分类问题。

虽然这是一个数据挖掘与机器学习中最基础的问题类型，并且在 Kaggle 上也已经有许多人进行了尝试，但是其依然存在一些难点和挑战，包括：

- 数据量小，模型在训练过程中**很容易出现过拟合**导致测试集上的准确率上不去
- 数据集中**存在较多的空缺项**，并且数据类型包括字符串 (name,ticket)，整数 (age,sibsp 等) 和浮点数 (fare)，**需要使用有效的预处理手段和特征工程**对数据进行良好的预处理才能取得较好的结果
- 传统的机器学习方法在 Titanic 问题上存在着性能瓶颈，需要探索更好的学习算法来预测生存结果。

因此，虽然 Titanic 这个题目相比于其他的题目而言并不算非常复杂，但是我认为这个数据集中依然有许多值得挖掘的东西，而模型层面也有更多可以选择的尝试，比起其他背景更加繁杂丰富的题目而言，我认为 Titanic 生存预测这一比赛更接近数据挖掘的本质，同时也依然有很大的研究空间，因此我选择本题作为数据挖掘导论课程的课程项目。

1.2 项目成果概述

本课程项目中，我进行了一系列的工作，并取得了如下成果：

- 完成了 Titanic 比赛，所得到的结果达到了 Kaggle 排行榜中 6% 的水平 (该比赛存在一些得分达到 100% 的作弊情况，因此实际排名应该更高，约为 5% 左右)
- 对 Titanic 数据集进行了一定的数据可视分析和预处理，提出了有效的数据预处理和特征工程策略
- 基于 Numpy 库从底层实现了 Decision Tree，Adaboost，随机森林，kNN 等机器学习算法，并应用在 Titanic 数据集中取得了比较好的效果
- 基于 PyTorch 库实现了含有 Dropout 机制的神经网络模型并应用在 Titanic 数据集中，取得了比较好的效果
- 基于 sklearn 机器学习库和 XGBoost 库调用一系列 baseline 算法应用在 Titanic 数据集中，并合自己实现的算法形成对照

1.3 项目提交文件目录

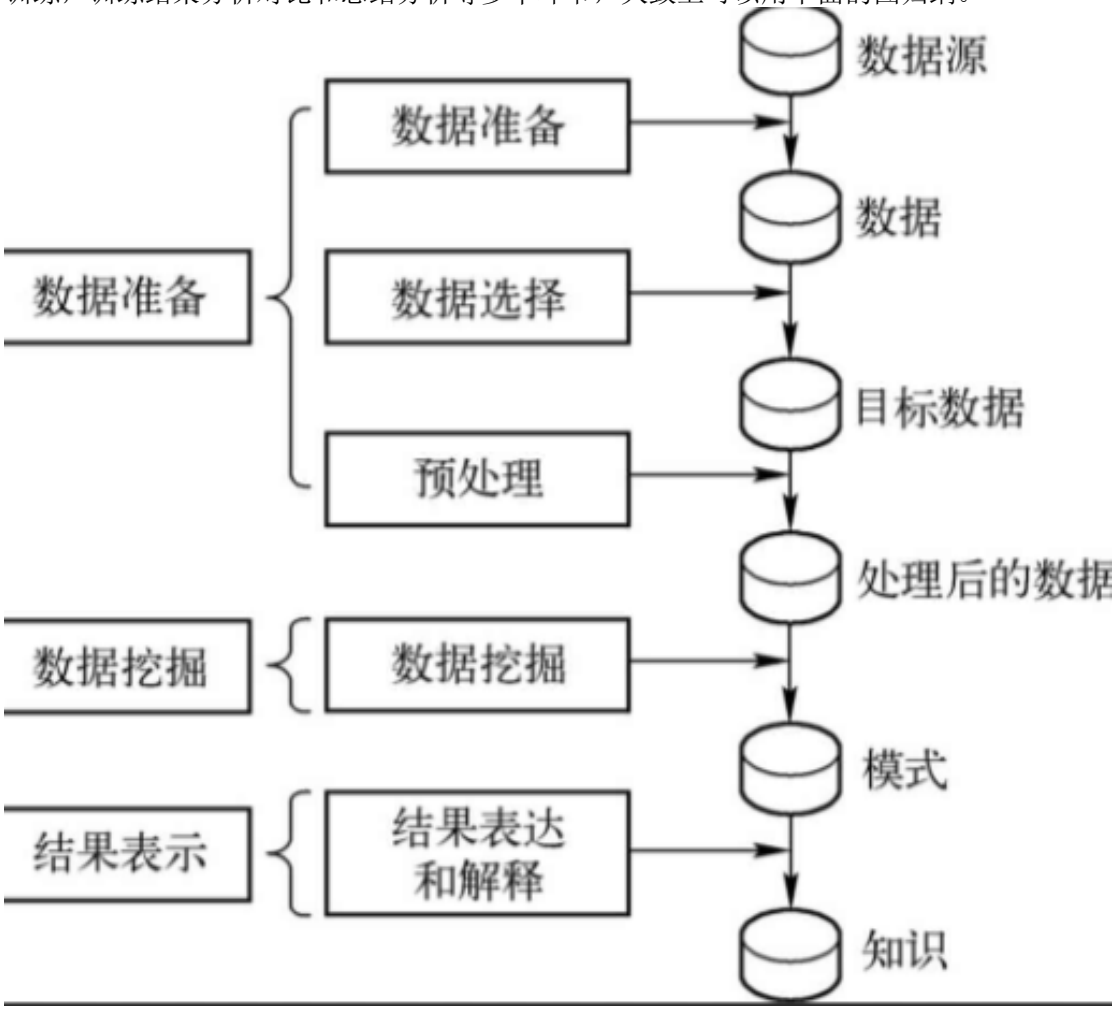
本项目最终提交的文件包括代码和文档，其中代码目录中包含了如下内容：

- Adaboost.py 自己实现的 AdaBoost 算法
- DecisionTree.py 自己实现的决策树算法
- KNearestMeighbor.py 自己实现的 kNN 算法
- model.py 自己使用 PyTorch 实现的神经网络模型
- RandomForest.py 自己实现的随机森林模型
- DecisionTree&RandomForest&Adaboost.ipynb 包含验证自己实现的决策树，随机森林和 AdaBoost 算法性能的代码
- ML-Baseline.ipynb 包含验证 sklearn 库中的传统机器学习算法性能的代码
- NN-Baseline.ipynb 包含验证 sklearn 库中的神经网络模型算法性能的代码
- NN-models.ipynb 包含验证 PyTorch 实现的神经网络模型算法性能的代码
- Titanic-XGBoost.ipynb 包含验证 XGBoost 算法性能的代码
- data2 处理后的数据集，包含训练集和测试集

二　研究方法与模型概述

2.1　项目总体流程

本次课程大作业中，我经历了数据分析，数据预处理 (数据补全和清洗)，模型构建和训练，训练结果分析对比和总结分析等多个环节，大致上可以用下面的图归纳。



2.2　数据集

Titanic 数据集包含 1300 余条数据集，训练集和测试集的比例约为 2：1，实际上数据集的规模是比较小的，本项目中对 Titanic 数据集进行了预处理，补全，清洗，可视化分析，特征工程等多个步骤的操作，最终得到了质量较高的训练数据集和测试数据集，具体的数据集处理将在第三部分展开。该数据集包含一系列泰坦尼克号乘客的基本信息，而目标则是预测乘客是否从泰坦尼克号事件中存活下来，测试集和训练集一共有 1309 条数据。该数据集主要有如下几个字段：

- Pclass 乘客所在的船舱等级
- Survived 乘客是否存活

- Name 乘客的名字
- Sex 乘客性别
- Age 乘客的年龄
- SibSp 乘客的兄弟姐妹和配偶数量
- Parch 乘客的父母与子女数量
- Ticket 乘客的票的编号
- Fare 票价
- Cabin 乘客的座位号
- Embarked 乘客的登船码头

2.3 算法和模型

本项目中使用了多种机器学习模型，从基于信息熵/信息增益比/基尼指数的决策树出发，使用集成学习模型中的随机森林，AdaBoost 和 GDBT，再到全连接神经网络和 Dropout 机制，我从底层实现了若干核心算法并将其应用到 Titanic 数据集上，同时选用了一系列线性模型作为 baseline 作为对照组，线性模型包括线性回归、逻辑回归、感知机、线性判别分析、kNN、支持向量机等等，以及朴素贝叶斯算法，这一部分将具体在第四部分展开说明。

2.4 代码运行环境

实验的基本环境如下：

- 操作系统：Windows 10
- 编程环境：Anaconda 4.10+Jupyter Notebook 4.4.0
- 编程语言：Python 3.6.5
- Python 库：Pandas，matplotlib, numpy, sklearn, PyTorch 等

三　数据集分析与处理

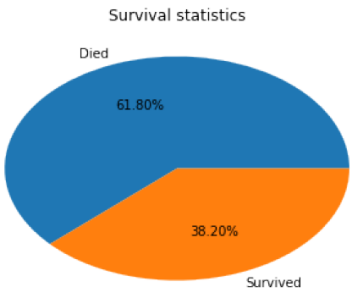
在使用机器学习算法构建模型之前，我先对数据集进行了一定的预处理并作出了一些可视化的分析。我们可以得到的初始数据如下：

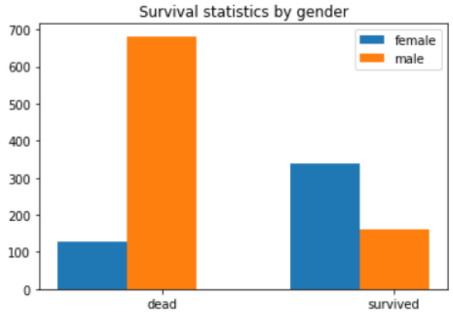
	Pclass	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	1	Hays, Miss. Margaret Bechstein	female	24.0	0	0	11767	83.1583	C54	C
1	3	0	Holm, Mr. John Fredrik Alexander	male	43.0	0	0	C 7075	6.4500	NaN	S
2	3	0	Hansen, Mr. Claus Peter	male	41.0	2	0	350026	14.1083	NaN	S

3.1　数据集可视化分析

首先我对数据集进行了简单的可视化分析操作，使用 matplotlib 等库进行统计图表的绘制，得到的一些结果有：

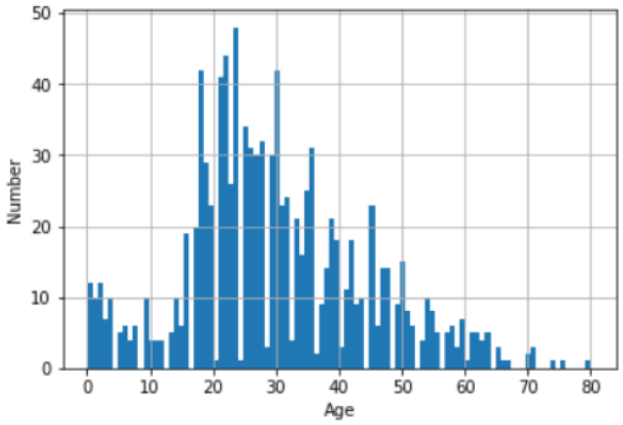
3.1.1　存活率统计





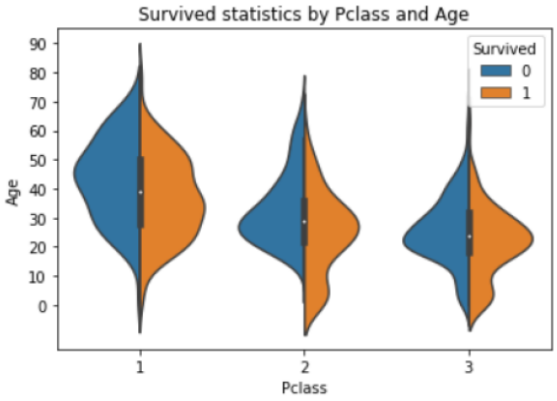
从按性别统计的存活率中计算得到，男性的存活率大约为 19% 而女性的存活率约为 72%，说明性别是影响存活率的一个非常重要的因素，女性生存下来的概率要比男性大很多。

3.1.2　乘客年龄分布



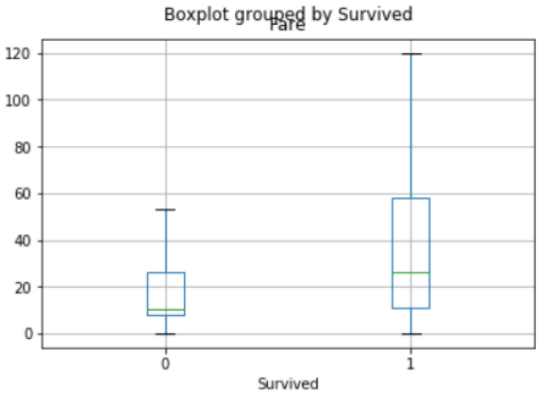
从年龄分布图中我们可以看出乘客的年龄段存在明显的分段结构，因此可以在数据预处理的时候对年龄进行分段。

3.1.3 存活率随年龄和船舱类型的分布



从这个图中我们可以分析出，相对而言，年轻乘客，特别是 10 岁及以下的儿童的存活概率更高，说明年龄也是影响存活与否的重要因素，同时船舱的不同也会影响不同年龄段乘客的存活概率。

3.1.4 票价与存活率



从这一结果中我们可以分析得到，买高价票的乘客存活概率高于低价票的乘客，从箱线图反映出的各种统计指标来看，存活的乘客的票价都比死亡的乘客的要高，我推测票价与消费水平和身份地位成一定的关系，身份地位高的人更容易活下来

3.2 预处理与特征工程

首先训练集和测试集的格式是不同的，训练集中的 Survived 标签是直接放在 train.csv 文件中的，而测试集中的存活情况被放在另外的一个提交文件夹中单独给出，为了方便数据的处理，我首先将测试集和 label 也进行了合并，在完成预处理之后再将其分割成 feature 和 label 两个部分，同时我结合数据集的一些特征为其构造了新的特征。

3.2.1 数据补全

对于每一列缺少的值采用一定的策略进行补全，比如 Embarked 全部使用 S 补全，而 Fare 使用中位数来代替，对于 Age 采用一个服从均匀分布的值进行代替。

3.2.2 数据转换

对一些难以直接分类的数据按照一定的规则进行处理使其可以用于补全，比如对于 Cabin，Sex 等特征采用 0 和 1 直接代替，对于 Embarked 特征使用 0，1，2 分别代表 S，C 和 Q，而对于 Fare 和 Age 等类型采用分段分类的方法构造特征。比如年龄以 16，32，48，64 为分界线分成四类 (事实上很多)

3.2.3 特征工程

在对数据的格式进行分析之后，我采取了一定的策略来构造新的特征，对原本的特征进行了一定的整合，包括：

- 家庭关系特征：Sibsp 和 Parch 都表示类似的特征，因此可以将 SibSp，Parch 合并成 FamilySize 特征来表示一个人的家庭情况
- 身份地位特征：数据中的人名可能暗示了人的身份地位，而身份地位和 Fare 等特征也有一定的关系，我们认为身份会影响到存活的情况，(事实上根据泰坦尼克号的传说，在逃生的时候所谓的贵族都将机会让了出来) 因此对乘客名字中暗含的身份进行了一定的提取，我们认为 Lady, Countess,Capt, Col,Don, Dr, Major, Rev, Sir 等标签往往代表着比较最贵的身份，因此最终将人的身份特征分成了 Mr，Master，Mrs，Miss 和 Rare 五类，分别使用 1-5 来表示类别。

3.2.4 预处理代码

实现了上述预处理手段和特征工程的代码如下所示：

```
full_data = [train, test]
# 数据预处理，包括去除缺失值，补全缺失值和非数值特征的映射
train['Has_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
test['Has_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
for dataset in full_data:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
for dataset in full_data:
    dataset['Embarked'] = dataset['Embarked'].fillna('S')
for dataset in full_data:
    dataset['Fare'] = dataset['Fare'].fillna(train['Fare'].median())

for dataset in full_data:
    age_avg = dataset['Age'].mean()
    age_std = dataset['Age'].std()
```

```
    age_null_count = dataset['Age'].isnull().sum()
    age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std, size=
        age_null_count)
    dataset.loc[np.isnan(dataset['Age']), 'Age'] = age_null_random_list
    dataset['Age'] = dataset['Age'].astype(int)

def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    if title_search:
        return title_search.group(1)
    return ""

for dataset in full_data:
    dataset['Title'] = dataset['Name'].apply(get_title)
for dataset in full_data:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr', 'Major
        ', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

for dataset in full_data:
    dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).astype(int)
    title_mapping = {"Mr": 1, "Master": 2, "Mrs": 3, "Miss": 4, "Rare": 5}
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare'] = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)
    dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] = 4
drop_elements = ['Name', 'Ticket', 'Cabin', 'SibSp']
train = train.drop(drop_elements, axis = 1)
test = test.drop(drop_elements, axis = 1)
```

四　机器学习算法和模型

4.1　Baseline 模型

为了体现项目中正式使用的模型与算法在 Titanic 数据集中的效果，我选择了一系列 baseline 模型，调用了 sklearn 机器学习算法库中的一系列标准算法，对于使用同样预处理手段的 Titanic 数据进行了 baseline 模型的实验，并获得了这些 baseline 模型在 Titanic 数据集上的预测准确率作为对照。

我具体选择的 baseline 模型有**线性回归，逻辑回归，感知机，朴素贝叶斯，线性判别分析，k 近邻算法和支持向量机**，下面对这些 baseline 模型进行简单的介绍。

4.1.1　线性回归

线性回归是最简单的一类机器学习模型，对于 n 维的样本 $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ 找到一系列线性函数 $\omega = [\omega_1, \omega_2, ..., \omega_n]$ 使得 $f(x) = \omega^T \boldsymbol{x} + b$

而在使用线性回归常见的 loss 函数定义最小平方误差 (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i, \omega))^2 \tag{1}$$

的情况下，线性回归模型的解为：

$$\omega = (XX^T)^{-1}Xy \tag{2}$$

本项目中，我们将预处理后的数据输入到 sklearn 的线性模型中并将生成的结果按照就近原则转化成离散的 0 和 1 来进行预测。

4.1.2　逻辑回归

逻辑回归是对线性回归的一种扩展，更加适合用于分类问题，逻辑回归往往选择通过一个 sigmod 函数将样本映射到某个区间上，以此来估计样本属于某种类别的概率从而达到分类的目的。我们经常选用的 sigmod 函数是：

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

比如对于二分类问题，可以将样本映射到区间 (-1,1) 上，此时如果计算结果为正数则说明样本属于正例，反之就是反例，可以表示为：

$$P(y_i = 1|x_i, \omega) = \sigma(\omega^T x_i) = \frac{1}{1 + e^{-\omega^T x_i}} \tag{4}$$

$$P(y_i = -1|x_i, \omega) = 1 - \sigma(\omega^T x_i) = \frac{1}{1 + e^{\omega^T x_i}} \tag{5}$$

逻辑回归往往使用随机梯度下降的方式进行优化求解，在实际的实验中我也发现效果相比线性回归要更好一点。

4.1.3 感知机

感知机模型是一种二分类的线性分类模型，输入 k 维线性空间中的向量，输出一个实例的类别 (正反类别分别用 +1 和-1 来表示)，可以将这个分类过程用一个函数来表示：

$$y = f(x) = \text{sign}(\omega x + b) \tag{6}$$

这里的参数 ω 和 b 就是感知机的模型参数，其中 ω，而实数 b 是一个偏置 (bias)，这里的 sign 函数是一个特殊的函数，当输入的 x 是正数或者 0 的时候函数值就是 1，输入的 x 是负数的时候函数值就是-1，实际上我们可以用超平面来理解感知机的概念，我认为感知机有 SVM 的雏形那种感觉，同时也有神经网络中的神经元的雏形。感知机一般使用随机梯度下降的方式进行求解，其损失函数被定义为错误分类的点的个数：

$$\min L(\omega, b) = -\sum_{x_i \in M} y_i(\omega x_i + b) \tag{7}$$

4.1.4 朴素贝叶斯

贝叶斯定理是一种非常经典且有效的分类手段，其核心是**通过先验概率和似然来计算后验概率**，即：

$$P(\omega_j|x) = \frac{P(x|\omega_j)P(\omega_j)}{P(x)} \tag{8}$$

$$P(x) = \sum_{j=1}^{c} P(x|\omega_j)P(\omega_j) \tag{9}$$

构建贝叶斯分类器的难点在于 $P(x|\omega_j)$ 难以估计，而朴素贝叶斯分类器的基本思想是，既然我们的困难是 $P(x|\omega_j)$ 涉及到 x 所有属性的联合概率不好估计，那我们就把联合概率的计算难度降到最低，也就是假设 x 的所有属性 (也可以叫做特征) 是互相独立的，此时对于 d 维的样本 $x \in D$, 贝叶斯的公式变成了

$$P(\omega|x) = \frac{P(\omega)P(x|\omega)}{P(x)} = \frac{P(\omega)}{P(x)} \prod_{i=1}^{d} P(x_i|\omega) \tag{10}$$

类似地，对于所有类别来说 P(x) 是相同的，因此朴素贝叶斯分类的目标就是

$$h_{nb}(x) = \arg\max_{c \in Y} P(\omega) \prod_{i=1}^{d} P(x_i|\omega) \tag{11}$$

训练集 D 中，令 D_c 表示第 c 类样本构成的集合，则类的先验概率

$$P(c) = \frac{|D_c|}{|D|} \tag{12}$$

对于离散的属性而言，可以令 D_{c,x_i} 表示 D_c 中第 i 个特征的取值为 x_i 的样本组成的集合，则条件概率可以估计为

$$P(x_i|\omega) = \frac{|D_{c,x_i}|}{|D_c|} \tag{13}$$

4.1.5 线性判别分析

线性判别分析 (Linear Disciminant Analysis) 是一种经典的降维方法，并且是一种线性的学习方法，LDA 的基本想法就是，对于给定的训练集，我们可以设法将这些样本投影到一条直线上，并且使得同类样本点的投影尽可能接近而不同类的样本点的投影尽可能远离，这样一来我们实际上就需要训练出一条直线来进行特征空间中的分类，而对于测试集中的数据，可以将其同样投影到这条直线上面去，再根据投影点位置来确定该样本属于哪一类别。

二分类问题的 LDA，可以假设问题定义在数据机 D 上，用 X_i, μ_i, Σ_i 分别表示某一类别的数据集合，均值和协方差矩阵，如果能够将数据投影到特征空间中的一条直线 ω 上，那么两类样本的中心在直线上的投影分别是 $\omega^T\mu_0$ 和 $\omega^T\mu_1$，如果将两类样本的所有点都投影到这条直线上面，那么两类样本的协方差是 $\omega^T\Sigma_i\omega$

而我们的目的是希望同类别的点在直线上的投影尽可能靠近而不同类的尽可能远离。因此可以让同类别的投影点的协方差尽可能小，即一个目标可以定义成：

$$\min\left(\omega^T\Sigma_0\omega + \omega^T\Sigma_1\omega\right) \tag{14}$$

而同时我们也希望不同类别的数据样本点的投影尽可能远离，因此可以让类中心之间的距离尽可能大，则另一个目标可以表示为：

$$\max||\omega^T\mu_0 - \omega^T\mu_1||_2^2 \tag{15}$$

因此线性判别分析的目标函数是：

$$\begin{aligned} J &= \frac{\left\|\omega^T\mu_0 - \omega^T\mu_1\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} = \frac{\left\|\left(\omega^T\mu_0 - \omega^T\mu_1\right)^T\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \\ &= \frac{\left\|\left(\mu_0-\mu_1\right)^T\omega\right\|_2^2}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} = \frac{\left[\left(\mu_0-\mu_1\right)^T\omega\right]^T\left(\mu_0-\mu_1\right)^T\omega}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \\ &= \frac{\omega^T\left(\mu_0-\mu_1\right)\left(\mu_0-\mu_1\right)^T\omega}{\omega^T\left(\Sigma_0+\Sigma_1\right)\omega} \end{aligned} \tag{16}$$

而实际上在二分类问题的情况下，线性判别分析和逻辑回归方法取得的效果几乎完全一致，可以认为是一种泛化的逻辑回归模型。

4.1.6 k 近邻

kNN：k-Nearest Neighbor 算法是一种分类算法，其核心想法是根据给定的训练集，对于需要判别的测试集中的样本，在训练集中找到与之最接近的 k 个样本，并统计这 k 个样本的分类情况，将出现次数最多的类别作为测试集中的样本的分类结果。如果将 k 个最近的样本用 $N_k(x)$ 来表示，那么 kNN 的分类决策用公式表示就是：

$$y = \arg\max \sum_{x_i \in N_k(x)} I(y_i = c_i) \tag{17}$$

kNN 的基本思路其实就是：如果一个东西看起来像鸭子，走路像鸭子，吃饭也像鸭子，那么它很可能就是一只鸭子。而” 最接近” 这个概念是有待商榷的，因为我们没有统一的距离度量法则，因此需要确定一种度量距离的方法。此外 k 也需要自己选择，k=1 的时候这个算法被称为最近邻算法

4.1.7 支持向量机

对于一个分类问题，我们如果可以找到一个向量 ω，使得对于数据集 D 中的任何样本 x，如果 x 是正例 (用 +1 表示) 就有 $\omega^T x > 0$，如果 x 是反例 (用-1 表示) 就有 $\omega^T x < 0$，那么我们就可以很好地对数据集进行分类，判断依据就是 $\omega^T x$

我们也可以这样来考虑，将数据集 D 中的每个样本 x 投射到 d 维的空间上，如果我们可以找到一个 d 维空间里的超平面 (hyperplane)$\omega^T x + b = 0$ 将这个空间划分成两个部分，其中一个部分里的样本 x 全是正例，另一个空间里的样本 x 全是反例，那么这个数据集的分类问题就解决了，但是实际情况并不会这么好，对于 d 维空间里的点 x，其到超平面的距离可以表示为：

$$r = \frac{|\omega^T x + b|}{||\omega||} \tag{18}$$

又为了能正确地进行分类，对于训练集中的数据，需要有：

$$\begin{cases} \omega^T x_i + b \leq -1, y_i = -1 \\ \omega^T x_i + b \geq +1, y_i = +1 \end{cases} \tag{19}$$

并且存在一些样本使得上面的等号成立，我们就称这些使得等号取到的点称为支持向量 (support machine)，每个支持向量到超平面的距离是：

$$r = \frac{1}{||\omega||} \tag{20}$$

则超平面两侧的两个支持向量到超平面的距离之和就是：

$$\gamma = \frac{2}{||\omega||} \tag{21}$$

这个量也被称为间隔 (margin)，为了使得分类效果尽可能地好，我们应该让这个间隔尽可能地大，因此我们的目标可以转化为求 $||\omega||^2$ 的最小值，即

$$\begin{aligned} &\min_{\omega,b} \frac{1}{2}||\omega||^2 \\ &s.t. y_i(\omega^T x_i + b) \geq 1 \end{aligned} \tag{22}$$

这个优化问题实际上就是支持向量机的基本形式。

4.2 项目使用的模型与算法

在上述提到的 Baseline 以外，我也使用了一系列比较复杂的机器学习模型作为正式的方法来解决 Titanic 的问题，包括决策树，随机森林，Adaboost，GDBT 以及神经网络，其中随机森林，Adaboost 和 GDBT 都属于集成学习的方法，神经网络属于深度学习的方法 (不过我的模型架构比较简单，只用了多层感知机和 dropout 机制)，并且**我对一些算法给出了自己的底层实现 (决策树、随机森林，Adaboost 和神经网络)**，不是仅仅调用 sklearn 中的算法库。

4.2.1 决策树

决策树是一种非常常见的分类模型，实际上是通过树形的结构来进行分类或者回归，在分类问题中可以根据样本的不同特征属性的不同属性值来进行多次分类的决策最终确定输入样本的类别，也就相当于执行了一连串嵌套的 if-else 语句，**也可以被认为是定义在特征空间和类空间上的条件概率分布**。决策树的优点是模型具有较好的可读性，分类的速度比较快。

决策树中的分类标准主要有信息熵、信息增益、信息增益比、基尼指数四种：

信息熵 Entropy 熵可以用来表示随机变量的**不确定性**。如果 X 是一个取值个数为有限个值即 $P(X = x_i) = p_i$，那么随机变量 X 的熵的定义就是：

$$H(x) = -\sum_{i=1}^{n} p_i \log p_i \in [0, \log n] \tag{23}$$

而对于随机变量 X 和 Y，如果 $P(X = x_i, Y = y_i) = p_{ij}$，那么在 X 给定的情况下随机变量 Y 的条件熵代表了 X 给定条件下 Y 的条件概率分布的熵对于 X 的数学期望：

$$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i) \tag{24}$$

当用数据估计得到熵和条件熵的时候，他们又分别被称为经验熵和经验条件熵。

信息增益 信息增益 (information gain) 表示**已经知道特征 X 的信息而使得类 Y 的信息的不确定度减少的程度**。特征 A 对于数据集 D 的信息增益 $g(D,A)$ 定义为 D 的经验熵和 D 在 A 给定条件下的经验条件熵的差值，即：

$$g(D,A) = H(D) - H(D|A) \tag{25}$$

信息增益又叫做**互信息 (mutual information)**，两个概念是等价的。基于信息增益的特征选择方法一般都是对于当前训练集 D，计算其每个特征的信息增益，并比较大小，选择信息增益最大的特征来进行当前层的分类。

假设训练的数据集是 D，有 K 个不同的分类 C_i 满足 $\sum_{i=1}^{K}|C_k| = |D|$，其中特征 A 有 n 个不同的取值 a_i，根据特征的取值将 D 划分成了 n 个不同的子集 D_i，而每个子集 D_{ik} 中表示属于类别 C_k 的样本，那么数据集 D 的经验熵可以表示为：

$$H(D) = -\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|} \tag{26}$$

特征 A 对于数据集 D 的条件经验熵可以表示为：

$$H(D|A) = \sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i) = -\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|} \tag{27}$$

信息增益比 以信息增益作为数据集划分依据的时候，容易偏向于选择取值更多的特征作为分类标准，但是使用信息增益比，可以校正这一偏差，信息增益比是当前数据集 D 关于特征 A 的信息增益和自身的熵的比值，即：

$$g_R(D,A) = \frac{g(D,A)}{H_A(D)} = \frac{g(D,A)}{-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\log_2\frac{|D_i|}{|D|}} \tag{28}$$

基尼指数 基尼指数是另一种决策树的评价标准，对于分类问题，假设样本集合中有 K 中不同的类别，每类出现的概率为 p_k 那么基于概率分布的基尼指数可以定义成

$$G(p) = \sum_{k=1}^{K}p_k(1-p_k) = 1 - \sum_{k=1}^{K}p_k^2 \tag{29}$$

而在样本集合 D 中，基尼指数可以定义为：

$$G(D) = 1 - \sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2 \tag{30}$$

我在决策树中实现了其中的三种分类标准 (信息熵、信息增益比和基尼指数)，同时实

现了一定的剪枝策略来减小过拟合对结果的影响。

4.2.2 随机森林

采用多个基学习器进行学习共同决策的模型被称为 ensemble 模型，这类机器学习方法也叫做集成学习方法，随机森林就是一种很常见的集成学习方法，通过训练一系列决策树来共同预测结果，并且随机森林是可以并行的。而在训练随机森林的过程中要特别注意的是将训练数据分成若干份，每次采用其中的一部分来训练一个决策树，这样才能训练出一系列各不相同的决策树。

4.2.3 Adaboost

AdaBoost，这是一种串行的 Ensemble 方法，通过若干次迭代的训练将训练数据学习成若干个弱分类器，然后通过调整权重来提高模型预测的准确度。

1. 将训练集 D 中的 N 个训练数据赋予初始的权重，一般来说是所有样本之间都相等，即

$$D_1 = (w_{11}, w_{12}, \ldots, w_{1N}) \quad w_{1i} = \frac{1}{N} \tag{31}$$

2. 对数据进行一共 M 轮的迭代，第 m 轮的时候训练数据的权重分布是 D_m，并训练得到一个弱分类器 $G_m(x)$，然后计算这个分类器的训练误差：

$$e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} \mathcal{I}(G_m(x_i) \neq y_i) \tag{32}$$

3. 然后计算该分类器的系数：

$$\alpha_m = \frac{1}{2} \log \frac{1 - e_m}{e_m} \tag{33}$$

4. 更新整个训练集的权重分布 $D_{m+1} = (w_{m+1,1}, w_{m+1,2}, \ldots, w_{m+1,N})$，其中 Z_m 叫做规范化因子，很明显这样的规范化因子使得前面的系数的和变成了 1，也就是说权重系数成为了一个概率分布

$$\begin{aligned} w_{m+1,i} &= \frac{\exp(-\alpha_m y_i G_m(x_i))}{Z_m} w_{m,i} \\ Z_m &= \sum_{i=1}^{N} \exp(-\alpha_m y_i G_m(x_i)) w_{m,i} \end{aligned} \tag{34}$$

5. 最后生成一个各个若分类器进行线性组合，形成权重不同的强分类器：

$$\begin{aligned} f(x) &= \sum_{i=1}^{M} \alpha_m G_m(x) \\ G(x) &= \text{sign}(f(x)) = \text{sign}(\sum_{i=1}^{M} \alpha_m G_m(x)) \end{aligned} \tag{35}$$

4.2.4 GDBT

GDBT 的全称是梯度提升树，实际上是一种使用回归树为基本分类器的提升方法，是一种简单的假发模型，即使用基函数的线性组合和前向分布算法，首先需要确定初始的提升树 $f_0(x) = 0$ 并且第 m 步的模型是：

$$f_m(x) = f_{m-1}(x) + T(x, \Theta_m) \tag{36}$$

这里的 $T(x, \Theta_m)$ 表示一颗决策树模型，通过经验风险的最小化来确定一棵决策树的参数，并且在优化过程中采用梯度下降法进行参数值的求解，其关键是利用损失函数的负梯度在当前模型的值作为提升树算法中残差的近似值来拟合一个回归树，这类算法被认为是统计学习中最有效的算法之一。

在具体的实验中我采用了 XGBoost 库中实现的一种名为极致梯度提升树 (Extreme Gradient Boost Tree) 的模型，它在 GDBT 的基础上进行了一定的优化，不仅支持并行训练，而且具有更好的泛化性能，不容易过拟合，具有比较强的鲁棒性。XGBoost（eXtreme Gradient Boosting）极致梯度提升，是基于 GBDT 的一种算法，这种算法相比于 GBDT 做出的改变包括：

- 利用二阶泰勒公式展开，优化损失函数
- 利用正则项，简化模型，避免过拟合
- 采用 Block 存储结构，支持并行计算

关于 XGBoost 具体的模型在 原论文 中有具体的阐述。

4.2.5 全连接神经网络

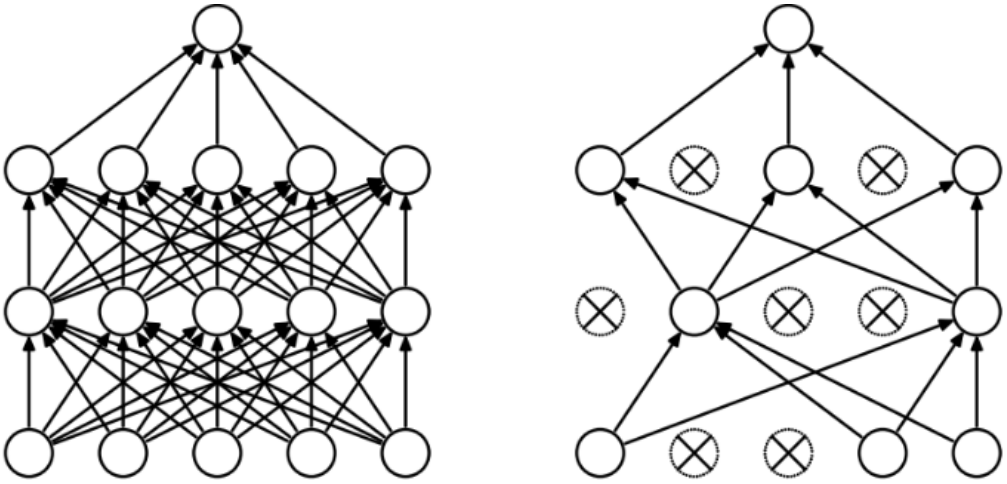
神经网络是深度学习中的非常重要的一个概念，本项目中使用的神经网络主要是最简单的全连接神经网络，由一系列全连接层 + 激活函数层组合而成，激活函数可以使用 logistic，tanh，relu 和不激活多种类型。这样的神经网络模型也叫做多层感知机，可以认为是多个感知机模型叠加起来组成的。

本次大作业中我对神经网络在 Titanic 中的应用方式做出了比较全面而细致的探究，包括激活函数的选取，模型架构的设计，训练方式和优化器的选取都做出了比较深入的探究，并在比较各种不同神经网络模型的基础上总结出了一些结论，也得到了比较好的结果。

4.2.6 基于 Dropout 的神经网络

DropOut 是一种在神经网络的训练中用的比较多的 trick，用来解决神经网络训练过程中的过拟合问题，DropOut 的基本想法就是设定一个 DropOut 的概率 p，当某一层的神经元被设定了这个概率 p 之后，就会出现以下两种情况：

- 训练的时候，对每个神经元而言，它有 p 的概率保持正常的工作状态，而有 $1-p$ 的概率“休息”，也就是不参与到神经网络的前向传播和反向传播过程中
- 测试的时候所有的神经元都参与到结果的预测中去



(a) Standard Neural Net　　(b) After applying dropout.

为什么 Dropout 会有效，这是一个论文中解释到，这是因为神经网络的训练本质上是将神经元训练出一种“协同作战”的能力，即神经元共同参与到最终的决策中，因此神经元之间的相互依赖性是比较强的，如果这个时候出现了一些表现不好的神经元，就会把所有的神经元带偏，并且随着放大效应逐渐累积。

而 DropOut 使得每次训练的过程中只有一部分神经元参与到训练中，并且每次参与的神经元的组合还很有可能不一样 (因为设定了概率 p)，这使得神经网络中的神经元形成了“小团队协作”的能力，增强了神经网络中单个神经元的预测能力 (因为训练的时候神经元个数减少意味着一个神经元需要负责学习更多的知识)，这样一来预测的准确度也就随之提高了。

本项目中也对使用了 Dropout 的神经网络模型进行了一定的探究，在使用 PyTorch 编写的神经网络基础上加入 Dropout 层，观察整个网络的性能。

五　实验结果与分析

5.1　实验设计说明

实验共分为两个部分，一部分是 Baseline 模型的在 Titanic 数据集上的表现，另一部分是实现的集成学习模型在 Titanic 数据集上的表现，二者互相作为对比，同时集成学习模型中将自己实现的决策树等算法和 sklearn 中的同类算法形成对照，最终得到的结果如下所示。

5.2　Baseline 模型训练结果

Baseline模型	测试集上的准确率%
线性回归	79.57
逻辑回归	80.28
感知机	75.77
朴素贝叶斯	76.00
线性判别分析	79.57
kNN	80.14
SVM	80.05
SVM(调参)	**81.95**
感知机(调参)	77.67

关于 Baseline 模型实验的几点说明：

- 所有模型均使用同版本的 sklearn 算法库，并使用默认的超参数
- 对于感知机和 SVM 等支持调参的

5.3　集成学习模型训练结果

5.3.1　决策树模型

我在自己实现的决策树中实现了三种分类标准，分别是 entropy，gini 和 information gain，得到的对比结果如下：

分类标准	测试集上的准确率
entropy	**82.63**
gini	82.54
infomation gain	82.16

我们发现在这个问题上，居然还是信息熵对样本的区分效果更好，在此基础上，我是用 entropy 作为标准，对决策树中设置的一些参数比如最大深度，最少采样节点数等进行了调节，

并尝试得到最优的结果，同时也对 sklearn 中的决策树分类器模型采取同样的操作进行调参，最终得到的对比结果如下：

模型	测试集上的准确率
决策树(base)	81.33
决策树(调参后)	**81.34**
决策树(sklearn)	80.04
决策树(sklearn调参后)	81.23

我们发现调参之后决策树的准确率有所提高，**同时我们自己实现的决策树的效果比 sklearn 库中自带的要略好一点**，这其中的原因也一度令我不解，但是查阅资料之后发现 sklearn 库中的决策树默认的分类标准是 gini 指数，而在本项目的场景下 gini 的表现并不如 entropy，或许这就是我自己编写的决策树的性能能比 sklearn 库中的略好的原因，同时 sklearn 中为了支持并行化的决策树训练使用了非常多的 trick，可能也导致性能有所损失。

5.3.2 随机森林与 Adaboost

随机森林和 AdaBoost 是两种非常常见的集成学习模型，本项目中实现了这两种算法，并将其和 sklearn 库中的随机森林分类器和 AdaBoost 进行了比较，值得注意的是 sklearn 中的 AdaBoost 并不能指定基学习器，因此性能上和自己实现的使用决策树作为基学习器的 AdaBoost 有所差异，得到的结果是：

模型	测试集上的准确率
随机森林(自己实现)	79.67
随机森林(sklearn)	**81.47**
AdaBoost(自己实现)	81.10
AdaBoost(sklearn)	80.28

事实上这里 **sklearn 中的随机森林和 AdaBoost 都没有使用效果最优的决策树作为基学习器**，因此性能反而不如自己实现的。

5.3.3 GDBT 与 XGBoost

本次实验中我使用 XGBoost，一个优化后的 GBDT 开源库来进行探索，同时 XGBoost 提供了大量的超参数可以调节，包括迭代次数，最大深度和最小 child 权重、采样概率、gamma 系数等等，在进行了一系列参数调节之后得到的结果如下 (具体的调参过程可以看提交的 notebook 文件，这里对于这类 dirty work 就不再赘述)：

模型	测试集上的准确率
XGBoost(Base)	80.76
XGBoost(调参后)	**82.18**

5.4 全连接神经网络训练结果

采用了 sklearn 中的 neural_network 中的 MLPClassifier，可以自由地构建不同的全连接神经网络，首先我选取了合适的激活函数和优化器，然后对模型的层级架构做出了一定的探索。

5.4.1 激活函数的选取

在其他条件相同的情况下对 4 中不同的激活函数都进行了测试，得到的结果如下：

激活函数选择	测试集上的准确率%
tanh	**81.24**
relu	80.52
identity	79.80
logistic	80.05

我们发现 relu 函数在该问题中的表现明显好于其他函数，这和 ReLU 函数的特性是有关系的，因此后续我们都将选用 ReLU 作为激活函数。

5.4.2 优化器的选择

sklearn 提供了 sgd，adam 和 lbfgs 等多种优化器来训练一个神经网络，实验的结果如下：

优化器类型	测试集上的准确率%
lbfgs	77.19
sgd	77.67
adam	**81.47**

- 虽然 lbfgs 的准确度相比于 sgd 而言较高，但是 lbfgs 是拟牛顿法的一种实现，这是一种二阶的梯度优化方法，因此速度相比于 adam 和 sgd 等一阶方法而言要慢很多
- adam 优化方法作为一种一阶的基于动量的优化方法性能表现非常不错
- 实际的实验中根据需要来选择优化器，根据效果表现选用 adam 作为优化器

5.4.3 模型的架构设计

在选定了优化器和激活函数之后我对隐层架构的设计进行了设计，首先我探究了单隐层的维数设计对准确率的影响，得到了如下结果 (均使用 ReLU 作为激活函数，lbfgs 作为优化器)：

单隐层维度	测试集上的准确率%
32	**82.66**
128	80.76
512	80.28
1024	80.76
2048	79.57

之后我们基于上面的结论尝试构建多层的全连接神经网络，并设计了隐层的结构，得到了如下结果 (此时均使用 ReLU 作为激活函数，adam 作为优化器)：

隐层设计	测试集上的准确率%
[512,128]	79.33
[512,128,16]	79.57
[512,128,16,4]	63.89
[512,1024,16]	80.28
[512,1024,64,16]	78.85
[512,1024,256,64,16]	**81.24**

5.5 含有 Dropout 机制的神经网络

我从训练的结果中发现了神经网络模型存在着比较严重的过拟合情况，测试集准确率相比于训练集的准确率要低了不少，而上面说到的 Dropout 机制正是神经网络训练过程中防止过拟合的办法，因此我在 model.py 中使用 PyTorch 编写了一个带有 Dropout 层的多层全连接神经网络，其架构如下：

```
class MLP(nn.Module):

    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(10, 512)
        self.fc2 = nn.Linear(512, 1024)
        self.fc3 = nn.Linear(1024, 64)
        self.fc4 = nn.Linear(64, 16)
        self.fc5 = nn.Linear(16, 2)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        out = F.relu(self.fc1(x))
        out = F.relu(self.fc2(out))
        out = self.dropout(out)
        out = F.relu(self.fc3(out))
        out = F.relu(self.fc4(out))
        return F.softmax(self.fc5(out))
```

最终得到的实验结果为：

模型架构	测试集上的准确率%
[512,1024,64,16]+Dropout	**83.52**

可以看到最终取得了本次实验中最好的效果。

5.6 实验结果总结

通过上述一系列的机器学习算法和模型训练的实验，我们可以得到如下结论：

- 实验最佳的结果约为 83.52%，达到了 Kaggle 排行榜上前 6% 的水平，如果将直接提交原数据的排名去掉或许会更高
- 集成学习模型采用共同决策和自适应权重分配等策略，使得模型的性能相比于传统的线性模型而言有了比较显著的提升
- 决策树模型在此类问题中的表现非常惊人，非常适合作为一种基学习器参与到集成学习中，但是决策树也存在过拟合的问题，可以使用控制最大深度等方式缓解，同时决策树也非常依赖于特征的构造和提取，也就是说要做好特征工程。
- 神经网络虽然可解释性差，但是在 Titanic 问题中也发挥了非常强的作用，分类的效果要比传统的机器学习方法略好一点，但是神经网络的训练需要依赖大量的数据，这样比较小的数据量非常容易引发过拟合的问题，而 dropout 一定程度上可以缓解这个问题，然而数据量终究太小，dropout 起到的作用也有限，我相信如果数据集规模更大的话，神经网络可以取得更好的效果，
- 同时神经网络的训练和构建也存在一定的随机成分，首先架构的确定没有较好的可解释性，此外训练的时候采用随机梯度下降为基础的方式使得收敛的情况难以预测，同一个模型训练效果也可能不同。

六　总结与展望

本次课程大作业的综合性较强，我在这个项目中体验到了数据挖掘的整个 pipeline，从数据的可视化分析和预处理开始，到底层实现自己的机器学习算法并使用处理后的数据构建模型，并和一系列 baseline 进行对比来验证自己提出的新模型的有效性，最终得到一个比较好的结果并分析出若干结论。虽然神经网络的准确率要比决策树以及传统的集成学习模型要高，但是其训练存在比较大的随机性，同时可解释性也远不如 GBDT 等统计机器学习的模型，因此我认为是否应该采用神经网络用在本项目中预测结果还有待商榷，但至少从结果来看，神经网络模型的效果略好于传统算法中最优秀的 GBDT

这一系列过程中我逐渐加深了对数据挖掘各个环节的理解，在动手实践的过程中我也踩了一些坑，总结出了一些经验和结论，我认为在整个过程中我的收获不仅是知道了什么算法，或者学会了怎么给模型调参数，而是建立起了一套完整的数据挖掘的世界观和方法论。