机器学习基础 实验六 实验报告

学号：202122202214

姓名：马贵亮

班级：2021级软件5班

实验目的

用集成方法对数据集进行分类

实验内容

利用若干算法，针对同一样本数据训练模型，使用投票机制，少数服从多数，用多数算法给出的结果当作最终的决策依据，对Titanic数据集进行分类，给出在测试集上的精确度。

除了投票法，其他的集成学习方法也可以。

实验来自kaggle入门赛 https://www.kaggle.com/c/titanic, 可以参考原网站代码与预处理部分，但与公开代码不同的在于，集成学习所用的基学习器需要自己实现而不能调用sklearn库。

数据集的分析是一个开放性问题，可以参考网站中的预处理方式。所选算法包括但不限于课堂上学习的模型例如： 决策树、SVM、KNN 、神经网络。

需要在网站上提交，不要求结果很高，但要求模型自己实现，如果有优化可以加分。

实验过程

由于该实验中会嵌入一些之前实验中已经实现的模型的基础，在之前实验中阐述过的模型在此不再阐述，具体的代码和模型的实现可以查看github：https://github.com/GuiliangMa/ExpML

1. 数据集（测试集+训练集）分析

考虑到测试集是透明的，那我们需要训练的模型实际是一个只需要结果的，仅对该数据有用的模型，因此可以在处理数据时同步处理train.csv和test.csv两个文件。首先对各列属性进行分析，如下表：

属性名	含义
PassengerId	乘客ID
Survived	是否生还（0=否，1=是）
Pclass	舱位等级（1=一等舱，2=二等舱，3=三等舱）
Name	乘客姓名 （姓-敬称-名）
Sex	性别
Age	年龄
SibSp	船上的兄弟姐妹/配偶数量
Parch	船上的父母/子女数量
Ticket	票号
Fare	票价
Cabin	船舱号
Embarked	登船港口（C=Cherbourg, Q=Queenstown, S=Southampton）

1.1 缺失值分析

训练集 (train.csv)

- `Age`：缺失177个值。
- `Cabin`：缺失687个值。
- `Embarked`：缺失2个值。

测试集 (test.csv)

- `Age`：缺失86个值。
- `Fare`：缺失1个值。
- `Cabin`：缺失327个值。

1.2 任务分析

考虑到我们的任务是对测试集的预测进行打榜，因此我们能看到所有的测试集，并且测试集对应的属性并不存在，那么为了提高数据处理的准确性和无误性，我考虑把测试数据和训练数据柔和在一起生成一系列的数据清洗处理规则，使得不会因为测试集中出现训练集中不存在的数据难以进行处理的情况，并且可以让测试集和训练集的数据处理后效果更加统一，对于某些特定的连续元素的归一化会更加的清晰，某些离散值的独热编码的数量也会减少，整体也可以提高模型预测的准确率。

1.3 数据填充

主要缺失的属性为 Age 属性和 Cabin 属性，而对于Embarked和Fare仅仅存在一个两个的缺失值，可以通过最简单的平均值填充或者。

对于 Embarked属性，可以采用众数进行填充，而对于Fare直接采用中位数进行填充。

再来考虑Cabin属性，Cabin属性的缺失性很高，**训练数据**有891项，共有687的空白项，占比**77%**；**测试数据**共418项目，共有327个空白值，占比**78%**。那么我把两者看作一个集体的情况下，共1309个样本，但是其中1014项为空白，占比**77%**。

再来看其非空的值，非空值的数据非常的杂乱，由于该信息为船舱号，其大致结构为 **大写字母+数字** 的组合，并且由于存在团购购票等问题，存在一栏中充斥着多个数据的情况。如果把多个数据进行分类归一，那么对于每一个样本都有着不同的船舱号。那么显然这种信息是冗余的，我将Cabin的大写字母进行提取，可更好的来泛化。当进行了提取之后，我们再来看这23%占比的非空数值。A：22、B：65、C：94、D：46、E：41、F：21、G：5、T：1，整体数据量都不大，如果用这样的分布去反向填充剩余的77%占比的数据，显然不合理。

因此在这里我考虑两种对这个数据的处理方法，第一种为：直接暴力将Cabin删除、第二种为：将所有未知的样本填充为U（一个毫不相关的标记）暂时采用第二种方法进行处理。

最后对于数据的填充只剩 Age属性，从缺失性上来看，训练数据共有891个样本，缺失177个值，缺失率占比19.8%；测试数据共有418个样本，缺失率占比为20%；总样本量为1309个样本，缺失263个样本，缺失率20%。缺失和非缺失占比约1：4，可以采用规则填充或者其他的机器学习方法进行预测。在随后的分析中再填充。

```
def Fill_Embarked_And_Fare_And_Cabin():
    data = pd.concat([train_data, test_data], ignore_index=True)
    train_data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
    test_data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)
    train_data['Fare'].fillna(data['Fare'].median(), inplace=True)
    test_data['Fare'].fillna(data['Fare'].median(), inplace=True)

    train_data['Cabin'] = train_data['Cabin'].apply(lambda x: x[0] if
pd.notna(x) else 'U')
    test_data['Cabin'] = test_data['Cabin'].apply(lambda x: x[0] if
pd.notna(x) else 'U')
```

1.4 数据修整

Name列处理

可以发现整个Name列大致可以分为三个部分，[姓 尊称 名]，考虑具体的情况，同样的姓氏可能是一家人，大致可能会购买同样的船票以及相似的位置，那么存活的可能性大致相似，与存活相关。而尊称中根据英语习惯有Mr. Mrs. Miss. 等，这些和年龄和婚否有关，由于年龄中存在缺失值，Miss.和Mrs.部分数据的情况下可以在一定情况下推测出大致的范围。因此将Name中的内容进行提取，提取Lastname和title，并将原数据列删除。

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


# 提取姓氏的函数
def extract_last_name(name):
    last_name_search = re.search('^(.+?),', name)
    if last_name_search:
        return last_name_search.group(1)
    return ""
```

Ticket列处理

Ticket列也是杂乱无章的，其数据的结构大致可以分为 票头（由大写字母开头，后续可能出现. / 字母 数字）+票号，部分样本的Ticket属性不存在票头。在这种情况下，我们先分别提取票头和票号，对于票号实际作用也不大，统计其对应的长度即可，对于不存在票头的样本，在票头填充Unknown。

而这样处理过后还没有结束，我们发现票头仍存在大量的异构数据例如 A/C，A./C，A./C.这样的数据，我们考虑到统计信息可能存在的偏差性，将其中的.和/字符进行删除，这样就可以把数据取值从51种降至37种。

```python
def extract_ticket_prefix(ticket):
    match = re.match(r'([A-Za-z][A-Za-z0-9./]*)', ticket)
    if match:
        prefix = match.group(1).strip()
        prefix = prefix.replace('.', '').replace('/', '')
        return prefix
    return 'Unknown'

def ticket_num_length(ticket):
    ticket_number = re.sub(r'^[A-Za-z][A-Za-z0-9./]*', '', ticket).strip()
    ticket_number = re.sub(r'\D', '', ticket_number)
    return len(ticket_number)
```

2. Age 回归填充

由于该部分在填充的过程中应该尽可能的保持更高的正确率，才能更好来实现对存活的预测。并且本实验要求的预测内容为`Survived`，并且要求在预测`Survived`的时候采用集成学习的方法，内部基学习器应当自己实现。Age作为数据处理的一部分，为了更加高效且准确的去处理后续的`Survived`，这在里我采用`sklearn`中的多个模型来对`Age`进行回归预测，通过查询相关资料和课程的学习共选择了12种模型，对其分别采用k-5折交叉验证，并且以负的均方误差作为评分标准，来选择一个最好的回归模型来对Age进行预测。

在此选择的模型有：线性回归、随机森林、Gradient Boosting、决策树、SVM、KNN、Ridge 回归、Lasso 回归、ElasticNet 回归、XGBoost、LightGBM、CatBoost和多层感知机。

在进行数据训练前首先需要对数据进行处理，考虑到测试集的透明性，将训练集和测试集进行合并，对其进行处理。对于连续值数据对其进行标准化、对于离散值的数据对其进行独热编码。

```python
train_df = pd.read_csv('../data/forAge/train.csv')
test_df = pd.read_csv('../data/forAge/test.csv')

```

```
test_passenger_ids = test_df['PassengerId']

combined_df = pd.concat([train_df, test_df], sort=False)

numerical_features = ['Pclass', 'SibSp', 'Parch', 'Fare']
categorical_features = ['Sex', 'Cabin', 'Embarked', 'Title', 'TicketPrefix']

numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

combined_processed = preprocessor.fit_transform(combined_df.drop(columns=
['PassengerId', 'Age', 'LastName']))
```

将数据进行简单的处理之后，需要先根据`Age`的有无筛选出对于Age的训练集和测试集。找到对应的训练集后，即可在上述模型下训练并求得得分最好的模型。

```
# 分离处理后的训练集和测试集
X_train_processed = combined_processed[:len(train_df)]
X_test_processed = combined_processed[len(train_df):]
y_train = train_df['Age']

print("Data is processed successfully\n")

# 定义模型
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Decision Tree': DecisionTreeRegressor(),
    'Support Vector Regression': SVR(),
    'K-Nearest Neighbors': KNeighborsRegressor(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'ElasticNet Regression': ElasticNet(),
    'XGBoost': xgb.XGBRegressor(),
    'LightGBM': lgb.LGBMRegressor(
        num_leaves=31,
        max_depth=-1,
        learning_rate=0.05,
        n_estimators=1000,
        min_data_in_leaf=20,
        feature_fraction=0.8,
        bagging_fraction=0.8,
        bagging_freq=5,
```

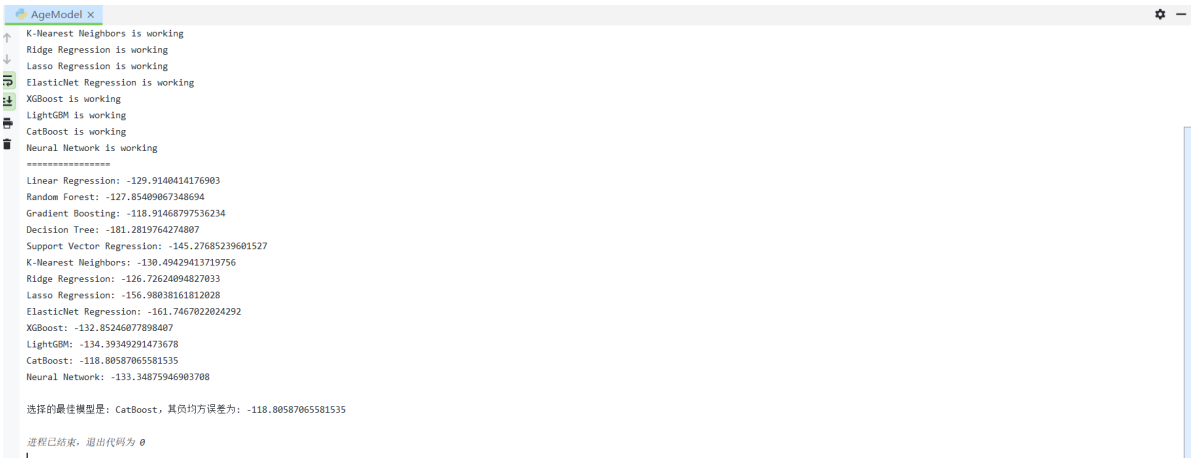
```
        verbose=-1
    ),
    'CatBoost': CatBoostRegressor(verbose=0),
    'Neural Network': MLPRegressor(max_iter=10000),
}

# 使用交叉验证评估每个模型
model_scores = {}
for name, model in models.items():
    print(f"{name} is working")
    scores = cross_val_score(model, X_train_processed, y_train, cv=5,
scoring='neg_mean_squared_error')
    model_scores[name] = scores.mean()

print("================")
# 输出每个模型的得分情况
for name, score in model_scores.items():
    print(f"{name}: {score}")

# 选择负均方误差（负值）最大的模型，即误差最小的模型
best_model_name = max(model_scores, key=model_scores.get)
best_model = models[best_model_name]
print(f"\n选择的最佳模型是: {best_model_name}，其负均方误差为:
{model_scores[best_model_name]}")

# 在整个训练集上训练最佳模型
best_model.fit(X_train_processed, y_train)
```



训练并选择好最好的模型后，需要将模型进行应用，用来预测没有年龄标签的样本数据，由于采用的是回归的模型，所以预测出的年龄的数据是一个不规则小数，这个不规则小数显然不应作为最后`Survived`预测的标签，而根据该网站对于数据的分析可知，对于不缺的年龄，原始数据种均采用x.5的方式表示，因此对于那些杂乱无章的小数，可以统一将其收束至x.5的格式，保留整数部分，小数部分填充为0.5，即可完成对年龄的预测。

```python
# 预测测试集的年龄
y_test_pred = best_model.predict(X_test_processed)

# 将预测结果保留整数部分，小数部分统一填充为0.5
y_test_pred_int = y_test_pred.astype(int) + 0.5

# 准备结果
results = pd.DataFrame({'PassengerId': test_passenger_ids, 'Predicted_Age':
y_test_pred_int})

print(results.head())

# 保存结果到CSV文件
results.to_csv(r'../data/forAge/predicted.csv', index=False)
```

得到的CSV文件大致如下：

	PassengerId	Predicted_Age
1	6	34.5
2	18	32.5
3	20	35.5
4	27	26.5
5	29	21.5
6	30	28.5
7	32	45.5
8	33	25.5
9	37	26.5
10	43	27.5
11	46	25.5
12	47	37.5
13	48	25.5
14	49	21.5
15	56	45.5
16	65	42.5
17	66	4.5
18	77	28.5
19	78	28.5
20	83	19.5
21	88	26.5

再将原先的数据中年龄不存在的样本，对照这个`predict.csv`进行填充。

```
predict_df = pd.read_csv('../data/forAge/predicted.csv')
predict_dict = pd.Series(predict_df['Predicted_Age'].values,
index=predict_df['PassengerId']).to_dict()
train_data['Age'] = train_data.apply(lambda row: fill_missing_age(row,
predict_dict), axis=1)
test_data['Age'] = test_data.apply(lambda row: fill_missing_age(row,
predict_dict), axis=1)
data['Age'] = data.apply(lambda row: fill_missing_age(row, predict_dict),
axis=1)
train_data.to_csv(train_processed_data_name, index=False)
test_data.to_csv(test_processed_data_name, index=False)
data.to_csv("../data/processed/data.csv", index=False)
```

填充后原始数据的初步处理就已经结束，所有缺失的值都已经被填充，所有异形的数据都已经被处理。

3. 标准化、独热编码、pca白化

3.1 标准化与独热编码

在进行实际训练之前，需要先对数据再次进行一次清洗处理，首先是要将连续的数据进行标准化，离散的数据采取独热编码的方式，这样进行处理后无论后续AdaBoost种选择哪一种模型，这个被清洗好的数据都可以很好的去直接去引用，减少后期选择不同模型之间的输入数据的二次处理。

标准化和独热编码的过程非常简单，即采用最朴素的标准差均值标准化和独热编码即可实现基本的功能：

```
train_data = pd.read_csv('../data/processed/train.csv')
test_data = pd.read_csv('../data/processed/test.csv')
data = pd.read_csv('../data/processed/data.csv')

numerical_features = ['Age', 'SibSp', 'Parch', 'Fare', 'TicketNumLen']
categorical_features = ['Pclass', 'Sex', 'Cabin', 'Embarked', 'Title',
'TicketPrefix']


def manual_one_hot_encode(df, column):
    unique_values = df[column].unique()
    for value in unique_values:
        df[f"{column}_{value}"] = df[column].apply(lambda x: 1 if x == value
else 0)
    df.drop(column, axis=1, inplace=True)

for num_feature in numerical_features:
    mean = data[num_feature].mean()
    std = data[num_feature].std()
    train_data[num_feature] = (train_data[num_feature] - mean) / std
    test_data[num_feature] = (test_data[num_feature] - mean) / std

for column in categorical_features:
    unique_values = data[column].unique()
    for unique in unique_values:
        train_data[f"{column}_{unique}"] = train_data[column].apply(lambda
x: 1 if x == unique else 0)
```

```
        test_data[f"{column}_{unique}"] = test_data[column].apply(lambda x: 1 if x == unique else 0)
    train_data = train_data.drop(column, axis=1)
    test_data = test_data.drop(column, axis=1)

train_data.to_csv('../data/preTrain/train.csv', index=False)
test_data.to_csv('../data/preTrain/test.csv', index=False)
```

采用这一段代码即可将之前处理好的数据进行标准化和白化，并且将该数据进行了保存，方便后续直接利用模型进行训练和预测。

3.2 PCA与白化（扩展）

PCA（主成分分析）

概念： PCA是一种降维技术，通过将数据投影到一个新的坐标系中，使得投影后的新坐标系中的每个坐标（即主成分）都是互相正交的，并且按方差从大到小排序。

意义:

- **降维**：减少数据集的维度，降低计算复杂度。
- **特征提取**：提取数据中的主要特征，保留数据中最重要的信息。
- **数据可视化**：将高维数据投影到2D或3D空间，便于可视化和理解。

具体作用:

- **消除冗余**：通过去除相关性高的特征来减少冗余信息。
- **降低过拟合风险**：减少特征数量，从而降低模型复杂度和过拟合风险。
- **提高计算效率**：减少数据维度，提升计算速度。

白化（Whitening）

概念： 白化是一种数据预处理技术，通过变换使得数据的每个特征具有单位方差，并且彼此之间不相关（即协方差为零）。

意义:

- **去相关**：消除特征之间的相关性，使得各个特征独立同分布。
- **标准化**：使数据具有相同的尺度（单位方差），有助于梯度下降等优化算法的稳定性。

具体作用:

- **提高算法稳定性**：白化后的数据有助于提高机器学习算法（特别是神经网络和梯度下降算法）的稳定性和收敛速度。
- **特征独立性**：消除特征之间的相关性，使得模型训练时每个特征独立，避免相关性引入的冗余信息。
- **数据标准化**：使得所有特征在相同尺度下，有助于模型的训练和参数优化。

3.3 实现

由于这部分为扩展内容并且为数据处理部分，因此我采用`sklearn`种的`StandardScaler`和`PCA`来实现主成分分析和白化的操作。

首先将若干不相干的属性进行剔除保存，随后将剩余的属性进行PCA+白化的操作，再将之前剔除的属性给补充回来，并且进行保存。

`pca`种的`n_components`设置为0.95，即可很明显观察到主成分提取的降维作用

```python
# 基于sklearn 的 PCA 包进行白化的操作
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

train_df = pd.read_csv('../data/preTrain/train.csv')
test_df = pd.read_csv('../data/preTrain/test.csv')
print(train_df.shape)
print(test_df.shape)

# 属性剔除... 省略

train_df['is_train'] = 1
test_df['is_train'] = 0

combined_df = pd.concat([train_df, test_df], ignore_index=True)
combined_data = combined_df.drop(columns=['is_train'])

scaler = StandardScaler()
combined_scaled = scaler.fit_transform(combined_data)
pca = PCA(whiten=True, n_components=0.95)
combined_pca = pca.fit_transform(combined_scaled)

combined_pca_df = pd.DataFrame(combined_pca, columns=[f'PC{i + 1}' for i in range(combined_pca.shape[1])])

# 重新加上标识符列
combined_pca_df['is_train'] = combined_df['is_train'].values

# 分离数据
train_pca_df = combined_pca_df[combined_pca_df['is_train'] == 1].drop(columns=['is_train'])
test_pca_df = combined_pca_df[combined_pca_df['is_train'] == 0].drop(columns=['is_train'])

# 属性恢复... 胜率

print(train_pca_df.shape)
print(test_pca_df.shape)

train_pca_df.to_csv('../data/preTrain/train_pca95.csv', index=False)
test_pca_df.to_csv('../data/preTrain/test_pca95.csv', index=False)
```

可以通过看见控制台的输出信息，来观测其降维的情况：

```
(891, 80)
(418, 79)
--------------------
(891, 64)
(418, 63)
```

可以说效果降维的效果还是非常显著。

4. AdaBoost 框架实现

4.1 理论支持

训练集为 $\mathbf{D}$

第 t 次的训练集权重为 $\mathbf{w}_t$，每次根据权重 $\mathbf{w}_t$ 对训练集 $\mathbf{D}$ 进行不放回采样获得本次的训练集 $\mathbf{D}_t$

用该次的训练数据 $\mathbf{D_t}$ 来训练本次的模型 $Model_t$，训练完所有的数据后得到判别函数 h_t，对整个训练数据集 $\mathbf{D}$ 计算带参数的误差 ϵ 和 α_t，并且根据 α_t 来计算新的 $\mathbf{w}_{t+1}$

$$h_t = \underset{h_j}{argmin} \quad \epsilon_j \qquad where \quad \epsilon_j = \sum_{i=1}^{m} \mathbf{w}_t(i)[y_i \neq h_j(\mathbf{x}_i)]$$

$$\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t}$$

$$\mathbf{w}_{t+1}(i) = \frac{\mathbf{w}_t(i) \cdot exp[-\alpha_t y_i h_t(\mathbf{x}_i)]}{Z}$$

$$result = sign[H(\mathbf{x}) = \sum_{i=1}^{T} \alpha_t h_t(\mathbf{x}_i)]$$

4.2 代码实现

```
class AdaBoost:
    def __init__(self, T):
        self.weight = None
        self.T = T
        self.modelList = []
        self.alphaList = []

    def pick(self, X, y, w):
        if isinstance(X, pd.DataFrame):
            # 获取索引数组并进行加权采样
            indices = np.random.choice(X.index, size=X.shape[0], p=w)
            # 使用iloc通过位置索引获取采样后的数据
            X_train = X.loc[indices].reset_index(drop=True)
            y_train = y.loc[indices].reset_index(drop=True)
        else:
            # 对于Numpy数组，直接进行加权采样
            indices = np.random.choice(range(X.shape[0]), size=X.shape[0],
p=w)
            X_train = X[indices]
            y_train = y[indices]
        return X_train, y_train

    def fit(self, X, y, model_type='Bayes'):
        self.weight = np.ones(X.shape[0]) / X.shape[0]
        self.model_type = model_type
        for index in range(self.T):
            X_train, y_train = self.pick(X, y, self.weight)

            # 模型嵌入并替换区域，此处的MODEL(args)只为一个代表，后续代码中会在此处进
行嵌入
            model = MODEL(args)
            model.fit(X_train,y_train,args)
            model.predict(X,args)

```

```
            y_temp = np.array(y)
            preds = np.array(preds)
            preds = np.sign(preds).astype(int)
            preds = np.where(preds == -1, 0, preds).reshape(y_temp.shape)
            error = np.sum(self.weight * (y_temp != preds))

            best_model = model
            best_error = error
            for models in self.modelList:
                preds = Model.predict(X,args)
                y_temp = np.array(y)
                preds = np.array(preds)
                preds = np.sign(preds).astype(int)
                preds = np.where(preds == -1, 0,
preds).reshape(y_temp.shape)
                error = np.sum(self.weight * (y_temp != preds))
                if error < best_error:
                    best_error = error
                    best_model = models

            error = best_error
            alpha = 1 / 2 * np.log((1 - error) / error)
            self.alphaList.append(alpha)

            preds = best_model.predict(X,args)

            y_temp = np.array(y)
            preds = np.array(preds)
            preds = np.sign(preds).astype(int)
            preds = np.where(preds == -1, 0, preds).reshape(y_temp.shape)
            error = np.sum(self.weight * (y_temp != preds))

            Z = np.sum(self.weight * np.exp(-alpha * y_temp * preds))
            self.weight = (self.weight * np.exp(-alpha * y_temp * preds)) /
Z


            self.modelList.append(best_model)
            print(f"T = {index + 1}, Training is Finished")

    def predict(self, X):
        res = np.zeros(X.shape[0])
        for index in range(self.T):
            model = self.modelList[index]
            alpha = self.alphaList[index]
            res += np.array(model.predict(X)) * alpha
        res = np.sign(res)
        res = np.where(res == -1, 0, res)
        return res
```

这样变初步完成了一个 AdaBoost 的框架，在这个框架基础上来调用不同的模型，即可实现AdaBoost的集成学习。

5. 内部嵌入不同的自己实现的模型

考虑到该网站一天有十次的提交上限，因此我从网络上找到一份声称100%正确率的数据集，在官网进行提交后发现准确率有99.7%以上，根据我初步计算，应该只有一个数据点保持不同，因此我在该数据集的基础上进行本地测试，并陆续将结果较好的模型组产生的数据上传到网站中。

并且由于挑选数据的过程具有随机性，因此下列所阐述的准确率有浮动是正常的。

5.1 嵌入贝叶斯决策器

首先第一个选取嵌入的决策器是贝叶斯决策器，这是我们在第一个实验种实现的决策器，整个先验概率和类条件概率密度采用计数的方式进行，并不引入其他的参数估计的方法。在上述给定的框架之中嵌入具体的代码如下。

```
# 模型引入
from Exp1.codes.NaiveBayesClassifier import NaiveBayesClassifier

# 嵌入部分
            if model_type == 'Bayes':
                y_train = y_train.replace(0, -1)
                model = NaiveBayesClassifier(alpha=1)
                model.fit(X_train, y_train)
```

这样进行嵌入后，模型即可正常运行。

首先采用没有进行pca白化的数据，并且考虑到进行pca与白化后，所有的数据变为方差为1的数据，这种情况下所有数值变化为连续的小数值，相对而言并不容易进行贝叶斯处理，因此这次嵌入不进行pca白化数据的对比。

当T=5时，其本地准确率可以达到76%，并且在网站上提交的准确率也可以达到76%

当T=10时：本地准确率会降低，大约会在72-73%附近

当 T = 20 时：本地准确率会进一步降低，大约会降至60-70%之间不等

5.2 嵌入KNN

第二个选择嵌入的模型是在实验二中实现的KNN模型，该模型相对简单，但是其预测过程和训练过程的参数与其他模型略有不同，因此在引进这个模型后仍需进行一定的处理。其固定窗口采用的是离该点最近的第n个点的距离所构成的超球体，并且用先验（计数）和类条件概率密度（knn估计）来进行判断所属的类别。

嵌入：

```
# 模型引入
from Exp2.codes.Part4.KNN import kNNClassifier

# 嵌入：
# 在fit中：
            if model_type == 'KNN':
                y_train = y_train.replace(0, -1)
                model = kNNClassifier(X_train, y_train)

            if model_type == 'KNN':
                preds = model.predict(X, 7)
```

```
            else:
                preds = model.predict(X)
# 在predict中
            elif self.model_type == 'KNN':
                res += np.array(model.predict(X, 7)) * alpha
```

这样进行嵌入后，模型即可正常运行。

KNN模型内部的度量方式选择欧式距离进行度量。由于knn对距离非常的敏感，而白化的过程会导致数据的分布形态发生改变，那么数据之间的距离也会发生改变，因此该模型也不适用与pca处理后的数据。

T = 5时，本地准确率有：74.7%，网站测试正确率也有74%以上

T = 10时，本地准确率有：65%左右

T = 20时，本地准确率有：63%左右，并且相当缓慢

5.3 嵌入决策树

第三个想嵌入的模型时实验三所实现的决策树模型，在这里我选取CART模型，该模型也相对简单，由于原先的模型是采用的全部连续值找最优分割中点的办法，因此在这个决策树中由于采用了独热编码，那么显然效果是可行的。

嵌入：

```
# 引入
from Exp3.codes.CART import CART

# fit嵌入：
            if model_type == 'CART':
                model = CART(max_depth=5, min_samples_split=5)
                model.fit(X_train, y_train, isPruned=True)
```

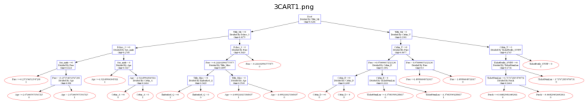
虽然并没有对白化的数据进行原数据映射，并且部分数据的分布更加离散，导致计算PCA后的数据会导致计算切分点的过程异常的繁琐，因此此步骤也不进行PCA后的数据分析。

采用原数据时：

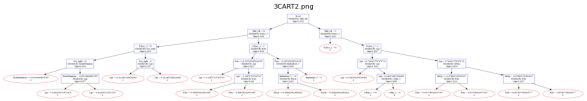
T = 3时：本地准确率有76%，其对应的决策树为：

3CART0.png

3CART1.png

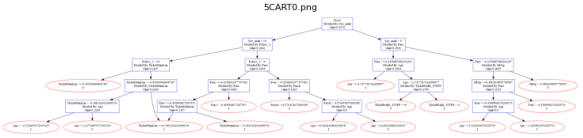


3CART2.png



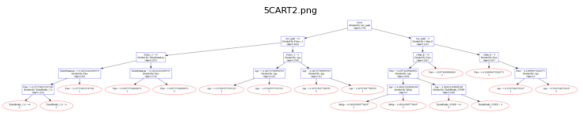
T = 5时：本地准确率有75%，其对应的决策树为：

5CART0.png

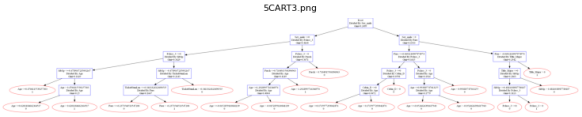


5CART1.png

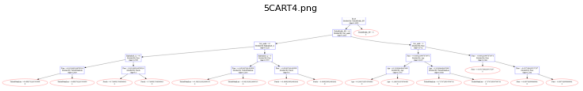
5CART2.png



5CART3.png



5CART4.png

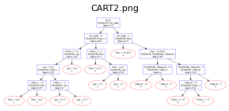


T = 10时：本地准确率有74%，其对应的决策树为：

CART0.png

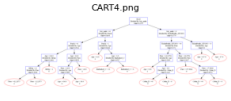
CART1.png

CART2.png



CART3.png

CART4.png



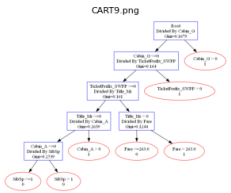
CART5.png

CART6.png

CART7.png

CART8.png

CART9.png



5.4 嵌入BP神经网络

第四个要嵌入的模型时实验四中实现的BP神经网络模型，设计参数为：

```
model = NeuralNetwork(layers=[X_train.shape[1], 2048, 512, 256, 128, 2],
                                    activations=['relu', 'relu', 'relu', 'relu', 'softmax'],
                                    dropout_rate=0,
                                    loss='cross_entropy')
model.train(X_train, y_train, 30, 32, 0.001, patience=5, decay_factor=0.1, task='Exp6',
                            val_size=0.1)
```

并将这些内容进行嵌入：

```
# 导入
from Exp4.codes.NNetwork import NeuralNetwork

# fit中嵌入：
            if model_type == 'BPNet':
                model = NeuralNetwork(layers=[X_train.shape[1], 2048, 512, 256, 128, 2],
                                      activations=['relu', 'relu', 'relu', 'relu', 'softmax'],
                                      dropout_rate=0,
                                      loss='cross_entropy')
                X_train = np.array(X_train)
                y_train = np.array(y_train).reshape(-1, 1)
                y_train = to_one_hot(y_train, 2)

                indices = np.arange(X_train.shape[0])
                np.random.shuffle(indices)
                X_train = X_train[indices]
                y_train = y_train[indices]

                model.train(X_train, y_train, 30, 32, 0.001, patience=5, decay_factor=0.1, task='Exp6',
                            val_size=0.1)
```

由于神经网络强大的实用性，因此可以同时分析pca前后不同的数据中进行学习。

因此分两部分阐述该过程的准确性

在原数据上时：

T = 3时：75-78%之间波动

T = 5 时：72-75%之间波动

T = 10时：72-75%之间波动

在pca白化上的数据时：

T = 3时：73-76%之间波动

T = 5时：70-76%之间波动

T = 10时：73-76%之间波动

在最终结果基本保持相似的情况下，采用pca白化的运行速度比正常的神经网络更快，降维还是可以体现效果。

5.5 嵌入SVM

第五个嵌入的模型时实验五中实现的支持向量机，在此直接采用实现的SVM中的fitstd策略来训练中间过程以此提高速度和效率，并且向量机和神经网络一样受距离影响小，因此可以将pca和非pca进行对比。

嵌入：

```
# 导入：
from Exp5.codes.SVM import SVM

# fit中嵌入：
            if model_type == 'SVM':
                y_train = y_train.replace(0, -1)
                model = SVM(C=10)
                model.fit(X_train, y_train, kernel='rbf', gamma=100)

# predict中嵌入：
            if self.model_type == 'SVM':
                res += np.sign(model.predict(X).reshape(res.shape)) * alpha
```

在原数据时：

T = 3时：60%左右，但验证集上的错误率很低，仅有15-18%这样的级别

T = 5时：60%左右，但是验证集上错误率很低

T = 10时：60%左右，验证集上错误类也大概在30-35%

采用PCA的数据后：

T = 3时：60%左右，验证集上的错误率只有15-20%这样的级别

T = 5时：60%左右，验证集上的错误率很低

T = 10时：60%左右，验证集上错误类也大概在30-35%

在最终结果基本保持相似的情况下，采用pca白化的运行速度比正常的神经网络更快，降维还是可以体现效果。

5.6 所有方法嵌入后的fit函数和predict函数

```
    def fit(self, X, y, model_type='Bayes'):
        self.weight = np.ones(X.shape[0]) / X.shape[0]
        self.model_type = model_type
        for index in range(self.T):
            X_train, y_train = self.pick(X, y, self.weight)
            if model_type == 'Bayes':
                y_train = y_train.replace(0, -1)
                model = NaiveBayesClassifier(alpha=1)
                model.fit(X_train, y_train)

            if model_type == 'BPNet':
```

```
                model = NeuralNetwork(layers=[X_train.shape[1], 2048, 512, 256, 128, 2],
                                      activations=['relu', 'relu', 'relu', 'relu', 'softmax'],
                                      dropout_rate=0,
                                      loss='cross_entropy')
                X_train = np.array(X_train)
                y_train = np.array(y_train).reshape(-1, 1)
                y_train = to_one_hot(y_train, 2)

                indices = np.arange(X_train.shape[0])
                np.random.shuffle(indices)
                X_train = X_train[indices]
                y_train = y_train[indices]

                model.train(X_train, y_train, 30, 32, 0.001, patience=5, decay_factor=0.1, task='Exp6',
                            val_size=0.1)

            if model_type == 'SVM':
                y_train = y_train.replace(0, -1)
                model = SVM(C=10)
                model.fit(X_train, y_train, kernel='rbf', gamma=100)

            if model_type == 'KNN':
                y_train = y_train.replace(0, -1)
                model = kNNClassifier(X_train, y_train)

            if model_type == 'CART':
                model = CART(max_depth=5, min_samples_split=5)
                model.fit(X_train, y_train, isPruned=True)

            if model_type == 'KNN':
                preds = model.predict(X, 7)
            else:
                preds = model.predict(X)

            y_temp = np.array(y)
            preds = np.array(preds)
            preds = np.sign(preds).astype(int)
            preds = np.where(preds == -1, 0, preds).reshape(y_temp.shape)
            error = np.sum(self.weight * (y_temp != preds))

            best_model = model
            best_error = error
            for models in self.modelList:
                if model_type == 'KNN':
                    preds = models.predict(X, 7)
                else:
                    preds = models.predict(X)
                y_temp = np.array(y)
                preds = np.array(preds)
                preds = np.sign(preds).astype(int)
                preds = np.where(preds == -1, 0, preds).reshape(y_temp.shape)
                error = np.sum(self.weight * (y_temp != preds))
                if error < best_error:
                    best_error = error
```

```
                    best_model = models

            error = best_error
            alpha = 1 / 2 * np.log((1 - error) / error)
            self.alphaList.append(alpha)
            if model_type == 'KNN':
                preds = best_model.predict(X, 7)
            else:
                preds = best_model.predict(X)

            y_temp = np.array(y)
            preds = np.array(preds)
            preds = np.sign(preds).astype(int)
            preds = np.where(preds == -1, 0, preds).reshape(y_temp.shape)
            error = np.sum(self.weight * (y_temp != preds))

            Z = np.sum(self.weight * np.exp(-alpha * y_temp * preds))
            self.weight = (self.weight * np.exp(-alpha * y_temp * preds)) /
Z

            self.modelList.append(best_model)
            print(f"T = {index + 1}, Training is Finished")

            if model_type == 'CART':
                dot = best_model.plot_tree()
                dot.render(f"../pic/CART/3CART{index}", format='png')

    def predict(self, X):
        res = np.zeros(X.shape[0])
        for index in range(self.T):
            model = self.modelList[index]
            alpha = self.alphaList[index]
            if self.model_type == 'SVM':
                res += np.sign(model.predict(X).reshape(res.shape)) * alpha
            elif self.model_type == 'KNN':
                res += np.array(model.predict(X, 7)) * alpha
            else:
                res += np.array(model.predict(X)) * alpha
        res = np.sign(res)
        res = np.where(res == -1, 0, res)
        return res
```

代码目录结构

```
/Exp6/
|------- code/
|       |------- AdaBoost.py 集成学习框架为AdaBoost内嵌入多种不同的基学习器策略，实
验预测主程序
|       |------- AgeModel.py 用于处理年龄回归问题的模型代码脚本
|       |------- compare.py 与从网上找到的疑似100%预测结果对比准确率的脚本
|       |------- pca_data.py 对两次处理完成的数据后的pca+白化操作
|       |------- PreDataProcessor.py 数据填充脚本
|       |------- PreTrainData.py 独热编码和标准化脚本
|       |------- plot.py b
|
|-------- data/
|       |------- answer/ 网上找到的疑似100%预测结果相关的数据
```

```
|       |------- forAge/ 与年龄回归相关的训练数据、测试数据、预测结果
|       |------- preTrain/ 经过了独热和标准化处理（或还有pca处理）的数据
|       |------- processed/ 经过了出Age外填充的数据
|       |------- submit/ 提交的csv
|       |------- titanic/ 实验的原始数据
|
|-------- pic/ 存储图像
|
|-------- 实验报告.md 实验报告Markdown
|
|-------- 实验报告.pdf 实验报告pdf
```

心得体会

本次实验通过对 Titanic 数据集进行数据处理和建模，深入探索了数据科学与机器学习的相关技术和方法。整个过程涉及数据清洗、特征工程、模型选择和评估等多个环节，每一步都需要仔细思考和不断尝试。

在实验初期，花费了大量时间对数据进行清洗和特征工程。通过提取姓名中的称谓和姓氏、处理票号等步骤，使得数据更加规范化。这个过程虽然繁琐，但对于后续建模有着至关重要的作用。缺失值的处理是数据预处理中的难点。对于年龄的填充，采用了多种回归模型进行预测，这不仅提高了数据的完整性，也为后续模型的准确性打下了基础。

在实验中，通过提取姓名中的称谓和姓氏、处理票号等方式创造性地构建了新的特征。这些特征在一定程度上反映了乘客的社会地位和经济状况，对预测乘客的生存概率有显著影响。对于类别变量，采用了独热编码进行处理，这种方法虽然会增加数据维度，但能使模型更好地理解这些离散特征。

本实验采用了多种机器学习模型，包括贝叶斯决策器、KNN、CART、支持向量机、神经网络等。集成于AdaBoost的集成学习方法，该方法通过组合多个基学习器，显著提升了模型的预测性能。集成学习的实现过程中，需要对每个基学习器的特点有深入了解，并根据具体问题选择合适的组合策略。

在模型调优过程中，发现不同模型对超参数的敏感度不同。通过不断调整超参数，找到最佳组合，使模型在测试集上的表现达到最优。尤其是在神经网络的训练过程中，学习率、迭代次数、批量大小等参数的调整对模型性能有显著影响。

通过本次实验，我不仅掌握了数据科学和机器学习的基本方法和技术，更深刻体会到实际数据分析中，数据预处理和特征工程的重要性。实验中的每一步都需要细致和耐心，同时也需要不断学习和尝试新方法。这次实验为我今后的数据分析工作奠定了坚实的基础。