课程大作业

学生姓名：李瑞, 杨益

学号： PB21000311,PB21000308

深度学习导论

授课教师：连德富

中国科学技术大学

June 11, 2023

Contents

深度学习导论课程大作业

1 引言

我们参加了 Kaggle 中的 Spaceship Titanic[1]比赛，在 2376 支队伍中取得了 0.82137 的分数，位列第五名的成绩 (**%0.21**)。数月参加比赛的亲身实践也促使我们重新思考了很多深度学习模型的重要问题，以下列出最重要的几个，我们也将在正文中给出我们的解决方案。

1. 如何为模型选出**有用的特征**?
2. 如何**选择**、**评估模型**并调整到合适的模型和超参数?
3. 在进行如比赛等相对长期工程中，应当如何制定合理任务规划、有序安排各项时间、保证模型的稳定复现和趋优，以达成系统的最佳?

2 数据集介绍

在这个比赛中，我们的任务是一个二分类问题，即预测太空船泰坦尼克号与时空异常相撞期间，乘客是否被传送到另一个维度，为了帮助预测，我们可以利用从船上损坏的计算机系统中恢复的一组个人记录。

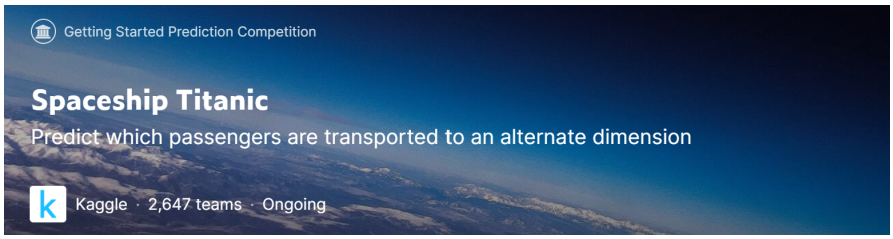

Figure 1: Spaceship Titanic Competition

3 数据分析与可视化

3.1 数据集描述

数据集 [2] 为训练集、测试的 csv 文件，训练集约占 $\frac{2}{3}$(约 8700 人)，测试集约占 $\frac{1}{3}$(约 4300 人), 最后的结果以 submission.csv 的形式上传，具体的数据 [3]如下所示。

- `PassengerId` - 字符串类型，每个乘客的唯一 ID：每个标识符的格式为 `gggg_pp`，其中 `gggg` 表示乘客所在的组，`pp` 表示组内的序号，组内的人员通常是家庭成员。
- `HomePlanet` - 字符串类型，固定几个字符串中的一个：乘客出发的星球，通常是他们的永久居住星球。

[1]Spaceship titanic 网址：https://www.kaggle.com/competitions/spaceship-titanic/overview
[2]Spaceship titanic 数据集网址：https://www.kaggle.com/competitions/spaceship-titanic/data
[3]Spaceship titanic 数据介绍网址:https://www.kaggle.com/code/gusthema/spaceship-titanic-with-tfdf/notebook

- `CryoSleep` - 布尔类型：表示乘客是否选择在航行期间进行悬浮动画休眠。处于悬浮动画休眠状态的乘客被限制在自己的舱房内。
- `Cabin` - 字符串类型，乘客所在的舱室号码：格式为 `deck/num/side`，其中 `side` 可以是 P（港口）或 S（右舷）。
- `Destination` - 字符串类型，固定几个字符串中的一个：乘客将要下船的星球。
- `Age` - 浮点数类型：乘客的年龄。
- `VIP` - 浮点数类型，乘客是否为特殊 VIP 服务支付费用。
- `RoomService, FoodCourt, ShoppingMall, Spa, VRDeck` - 浮点数类型，乘客在泰坦尼克号上各种豪华设施上的消费金额。
- `Name` - 字符串类型，乘客的姓和名。
- `Transported` - 布尔类型：乘客是否被传送到另一个维度。这是需要预测的部分。

我们首先绘制了所有特征相关性的 Heatmap，关键代码和图如下所示。

```
def correlation_heatmap(df):
_ , ax = plt.subplots(figsize =(14, 12))
colormap = sns.diverging_palette(220, 10, as_cmap = True)

_ = sns.heatmap(
    df.corr(),
    cmap = colormap,
    square=True,
    cbar_kws={'shrink':.9 },
    ax=ax,
    annot=True,
    linewidths=0.1,vmax=1.0, linecolor='white',
    annot_kws={'fontsize': 5 }
)
plt.title('Pearson Correlation of Features', y=1.05, size=15)
correlation_heatmap(train)
```

数据分析和可视化可以帮助我们深入了解数据集的结构、特征和分布。通过分析数据的统计特征、摘要信息和可视化图表，我们可以获得对数据的整体认识，发现数据中的模式、异常和趋势，帮助我们选择、加工原始特征。

所有数据分析和可视化代码整理在 *src* 目录下数据分析和可视化.py 中

3.2 数据缺失值

我们首先使用 *.isna*() 统计数据缺失值，核心代码和统计结果如下。

```
#统计数据缺失值
print('Missing values')
print('Train Set')
print(train.isna().sum())
print('Test set')
print(test.isna().sum())
```

缺失值约占全体特征的%2, 所占比例并不高，但是从图可以看到，有%25 的用户至少缺少了一个值，我们也绘制了缺失值的 Heatmap，

给定特征可以被分为六个连续型值的特征，七个离散型的特征 (其中包含二分类和固定的字符串)，对于不同的特征类型，我们需要进一步根据数据特点并观察数据分布，选择合适的填充方法。

```
train.dtypes    #特征类型
```

由于为二分类任务，我们绘制饼状图调研训练集各类的分布情况，并观察各个标签下主要特征的分布情况。

```
plt.figure(figsize=(6,6)) #训练集标签分布情况
train['Transported'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=True, textprops={'
```

3.3 对于连续型特征的分析

```
plt.figure(figsize=(10, 4))#对连续型特征的分析
sns.histplot(data=train[train['Transported']=='True'], x='Age', bins=30, kde=True, label='Survived', stat="d
sns.histplot(data=train[train['Transported']=='False'], x='Age', bins=30, kde=True, label='Not Survived', st
plt.title('Age Distribution')
plt.xlabel('Age (years)')
plt.legend()
plt.show()
```

可以看到，True 和 False 的标签量相对均衡，而且我们进一步研究了标签与 Age 和 Cost(两个 float 特征) 的影响，我们发现年龄分布较为均衡，但是开销分布非常不均衡，花销从 0 取到 14327，但实际上大于 2000 的只占 2%。

通过观察数据，我们得出了以下几条结论

- 未成年 (0-18) 更可能被传送
- 中年 (18-60) 更可能不被传送
- 老年基本上是等概率的

从观察数据出发，在数据预处理中我们可以进一步把年份分组创建一个新的特征，便于机器学习学习；我们进一步把年龄分组，得到六祖年龄，如图所示

对于开销，

- 大部分人没有花任何钱，并且随着开销以类似指数分布的形式递减，
- 被传送的人开销往往更少
- FoodCourt ShoppingMall 和其他三项分布不一致，可以视为不同层次的生活需求

并且从观察数据出发，在数据预处理中我们可以用总价格来表示这五个特征，并且可以用正则化的方式例如 log 来处理分布不均衡的问题。

3.4 对于离散型特征的分析

```
#对分类型特征的分析
cat_feats=['HomePlanet', 'CryoSleep', 'Destination', 'VIP']
fig=plt.figure(figsize=(10,16))
for i, var_name in enumerate(cat_feats):
    ax=fig.add_subplot(4,1,i+1)
    sns.countplot(data=train, x=var_name, axes=ax, hue='Transported')
    ax.set_title(var_name)
fig.tight_layout()  # Improves appearance a bit
plt.show()
```

这些分类的特征通常是二分类或者三分类，因此我们可以通过比较简单的可视化进行观察。直观来说，一个重要的特征应当是在不同标签上分布不均衡的，因此我们通过以下代码绘制直方图并且观察到以下的结论。

- VIP False 和 True 的分布很不均衡，并且 VIP、HomePlanet 和 Destination 在 transported 上没有明显差别，因此可能不是一个很好的特征
- CryoSleep 或许是一个很好的特征选择，因为他分布较为不均衡。

从观察数据角度出发，我们可以考虑扔掉 VIP 这个特征以防止模型过拟合。

对于特征 `gggg_pp`，其中 `gggg` 表示乘客所在的组，`pp` 表示组内的序号，组内的人员通常是家庭成员，`Cabin` - 字符串类型，乘客所在的舱室号码：格式为 `deck/num/side`，其中 `side` 可以是 P（港口）或 S（右舷）。。我们可以考虑将字符串分割开并且对不同的字符串取哈希值，将字符串预处理为简单标签以便于后续处理，其中关键代码如下。

```
#extract ID and hash
train['Group'] = train['PassengerId'].apply(lambda x: x.split('_')[0]).astype(int)
test['Group'] = test['PassengerId'].apply(lambda x: x.split('_')[0]).astype(int)
train['Group_size']=train['Group'].map(lambda x: pd.concat([train['Group'], test['Group']]).value_counts()[x])
test['Group_size']=test['Group'].map(lambda x: pd.concat([train['Group'], test['Group']]).value_counts()[x])
```

我们统计了所有组和家庭的规模，如图所示，我们发现组规模近似以指数方式衰减，但家庭人数一共有 6217 种，而且除了单独出行以外全是被传送的概率更高，因此我们可以考虑额外指定一个标签，判断一个人是否是一个人出行。

3.5 数据分析的总结

综上，我们进行了数据分析和可视化，数据分析和可视化可以帮助我们评估不同特征的重要性和相关性，从而指导特征选择和特征工程的过程。通过可视化展示数据的分布和相关性，我们可以更好地理解数据，并选择和创建最相关和有用的特征。具体而言我们发现了数据种的异常缺失值，并通过可视化和统计分析，进行了检测和识别异常值、缺失值、重复值等数据质量问题，这有助于后续清洗和处理数据，提高数据的准确性和可靠性；我们分别研究了不同种的数据类型，发现不同数据之间数据的模式和趋势，对特征进行探索性分析的过程种，我们发现了数据的一些内在的模式、趋势和关联性。这有助于我们理解数据集的特征、属性和潜在规律，再次基础上，我们进行数据预处理，把 csv 数据处理为 numpy 表示的 embedding，便于我们后续调取训练。

4 数据预处理

在数据分析与可视化的基础上，我们对原始数据进行预处理以便于模型训练，包括了缺失值填充、特征转换、独热编码和数据正则化等部分。

我们先读取数据、去掉标签并把字符串类型特征分割开

```
pd.set_option('display.max_columns', None)
df = pd.read_csv("titanic/train.csv")
y = df.pop("Transported") #transported弹掉

def CabinSplit(df):
    df["Deck"] = df["Cabin"].str.split("/").str[0]
    df["Num"] = list(map(float, df["Cabin"].str.split("/").str[1]))
    df["Side"] = df["Cabin"].str.split("/").str[2]
    df = df.drop(["Cabin", "PassengerId", "Destination", "Name"], axis=1)
    return df
df = CabinSplit(df)
```

4.1 缺失值填充

均值填充、零填充、众数填充均是比较常见的缺失值填充方式，根据缺失值的特点和数据的属性来选择最合适的填充策略。以下是核心代码和我们对各个特征选取的填充方式与理由。

- 对于连续型变量”Age” 采取均值填充，使用变量的均值来填充缺失值，这是由于我们希望年龄能够尽量保持数据的整体分布特征。而且年龄分布相对比较均衡和对称，没有明显的长尾效应。
- 对于所有开销”RoomService”, ”FoodCourt”, ”ShoppingMall”, ”Spa”, ”VRDeck”, ”Num” 采用零填充，这是由于在之前数据分析和可视化的过程中，我们发现开销的大部分均为 0，因此我们也选择把缺失值置为 0.
- 对于离散型特征”HomePlanet”, ”CryoSleep”, ”VIP”, ”Side”, ”Deck” 采用众数填充，众数填充可以在一定程度上保持原始数据的分布特征，并且对于离散型变量而言，众数相对比较有效。

核心代码为

```
def FillNaN(df):
    mf = ["HomePlanet", "CryoSleep", "VIP", "Side", "Deck"]
    mean = ["Age"]
    zero = ["RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck", "Num"]
    df[mf] = df[mf].fillna(df.mode().iloc[0]) #1.众数填充
    # print(type(df.mode().iloc[0]))
    mode = np.array(len(df.mode().iloc[0]))
    mode = df.mode().iloc[0]
    np.save("processed/comple.npy",mode)
    df[mean] = df[mean].fillna(df[mean].mean())
    df[zero] = df[zero].fillna(0)

    # print(mode,df.mode().iloc[0],df[mean].mean()) #Age    28.82793
    return df
df = FillNaN(df)
```

4.2 特征转换与创建新特征

由之前的数据分析，我们把所有的开销"RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck" 加起来以一个新特征"MoneySpent" 作为表示，而将之前的特征抛弃，避免模型过拟合；我们把年龄分为四组，并以 1-4 进行编码，核心代码如下。

```
def FeatureCreation(df):
    df["MoneySpent"] = df["RoomService"] + df["FoodCourt"] + df["ShoppingMall"] + df["Spa"] + df["VRDeck"]
    df.insert(loc = 3, column="AgeCategories",value=0)
    df.loc[df["Age"] <= 14, "AgeCategories"] = 1
    df.loc[(df["Age"] > 14) & (df["Age"] <= 24), "AgeCategories"] = 2
    df.loc[(df["Age"] > 24) & (df["Age"] <= 64), "AgeCategories"] = 3
    df.loc[(df["Age"] > 64), "AgeCategories"] = 4
    return df
df = FeatureCreation(df)
```

4.3 对特征 one-hot 编码

我们将分类变量进行独热编码或标签编码，以便后续在机器学习模型中使用。我们使用 $pandas.get_dummies$ 方法对 df 中的"Deck"、"Side"、"AgeCategories" 和"HomePlanet" 这四个分类变量进行独热编码，并将编码结果与原数据框 df 利用 $pandas.concat$ 进行拼接；我们使用 $sklearn.LabelEncoder().fit_transform$ 方法对"CryoSleep" 和"VIP" 这两个二元变量进行标签编码（Label Encoding），将分类变量转换为数值型。

```
def Encode(df):
    df = pd.concat([df, pd.get_dummies(df[["Deck", "Side", "HomePlanet"]])], axis=1)
    df = pd.concat([df, pd.get_dummies(df["AgeCategories"])], axis=1)
    df[["CryoSleep", "VIP"]] = df[["CryoSleep", "VIP"]].apply(LabelEncoder().fit_transform)
    df = df.drop(["Deck", "Side", "HomePlanet", "AgeCategories"], axis=1)
    return df
df = Encode(df)
y = LabelEncoder().fit_transform(y)
```

4.4 保存训练集标签

注意到我们一开始将训练集中的标签 "Transported" 弹出丢弃，在最后我们取出标签存在一个 numpy 数组中进行保存，核心代码如下。

```
    df1 = pd.read_csv("titanic/train.csv")
    score = np.zeros((df.shape[0],1),dtype=int)
    score1 = np.zeros((df.shape[0],1),dtype=int)
    score = df1["Transported"]
    score1[score == False] = 0
    score1[score == True] =  1
    # print(score1[1:5])
    np.save("processed/score.npy",score1)
```

4.5 特征缩放

最后，我们采取机器学习常用的特征缩放方法，它对特征数据进行尺度调整，使得不同特征具有相似的数值范围。可以消除特征之间的量纲差异，以确保模型能够准确、稳定地学

习和预测，能够提高模型性能，加快模型收敛，提高模型的鲁棒性，使得模型表现的更佳，具体代码如下。

```
from sklearn.preprocessing import StandardScaler

train_scale = train
print(test_scale.shape,train_scale.shape)

scaler = StandardScaler()
train_scale = scaler.fit_transform(train_scale)

test_scale = scaler.transform(test_scale)
print(test_scale.shape,train_scale.shape)

np.save("processed/train_scaled.npy",train_scale)
np.save("processed/test_scaled.npy",test_scale)
```

我们创建一个 StandardScaler 对象 scaler，用于对数据进行特征缩放。StandardScaler 将数据缩放到均值为 0，方差为 1 的标准正态分布。接下来，对训练集数据进行特征缩放处理。通过调用 $scaler.fit_transform(train_scale)$，将训练集数据传递给 $fit_transform()$ 方法进行拟合和缩放处理，使得训练集数据按照指定的缩放方式进行转换。然后，对测试集数据进行特征缩放处理。通过调用 $scaler.transform(test_scale)$，将测试集数据传递给 transform() 方法进行缩放处理，使用与训练集相同的缩放方式进行转换。最后，使用 np.save() 函数将经过特征缩放的训练集数据和测试集数据保存到新的文件中总体而言，这段代码的目的是将训练集和测试集的数据进行特征缩放处理，并将缩放后的数据保存到新的文件中，以便在后续的模型训练和评估中使用。

4.6 数据预处理总结

我们对原始数据进行了数据预处理，并将原始数据加工为 *processed* 目录下三个.npy 的 numpy 数组，在后续模型训练中，我们可以直接读取 numpy 并输入模型中进行处理。可以看到，对数据作合适的预处理并不是一件很简单的事情，这一切应该建立在对数据充足的分析和可视化的基础上。

5 模型选择和训练

在之前的部分里面我们得到了一个相对满意的特征表示，下面我们开始挑选合适的机器学习模型。我们在 Kaggle 平台了解开源的高分模型时，惊奇的发现基本为传统机器学习算法，并且在任务开始的初期，我们也尝试使用了深度学习方法进行训练，效果却不尽人意，这促使我们开始思考这其中的原因与如何根据数据集和任务选择合适的模型。

5.1 机器学习模型 XGboost

最佳的开源模型[4]采用传统机器学习模型并取得了 top2% 的结果，相比下开源的深度学习神经网络模型最佳也仅在 Top6% 左右。XGBoost 基于梯度提升树（Gradient Boosting Tree）的框架，通过迭代地训练多个决策树，并将它们组合成一个强大的集成模型。每一棵树都是通过优化损失函数来最小化预测值与真实值之间的误差。算法通过使用梯度信息来对损失函数进行优化，这使得 XGBoost 能够更好地适应数据中的噪声和复杂关系。

XGBoost 的特点之一是它的灵活性和**可扩展性**。它支持各种损失函数和目标任务，如分类、回归和排序。此外，XGBoost 还提供了许多优化技术，如正则化、学习率衰减和并行计算，以加快训练速度和提高模型性能。

5.2 如何选择选择的模型

选择一个模型，首先我们应当考虑模型的 “capacity”（没有找到合适的翻译，便以此称呼）[5]。

可以看到，比较小的 capacity 往往会欠拟合，而比较大的 capacity 很容易过拟合，因此我们要考虑到任务的复杂度和数据集的大小，选择合适的深度学习模型。注意到模型的特征仅有几十个且训练样本也在 9000 个左右，数据集相对较小，其实相当容易过拟合，因此我们作出以下假设并且进行了一些初步的实验以进行验证。

1. 前馈、CNN 等神经网络可扩展性相对较差，而其本身 “capacity” 比较大，在没有仔细选择超参数和训练策略的前提下，神经网络模型容易在数据集上过拟合，从而很难取得很高的效果

由于我们的目的是搭建一个优秀的模型，而这一部分仅仅是为我们提供模型选择的方向。我们设计了一个简单的实验来，并只调整了如学习率等必要的超参数，通过对比神经网络，CNN，XGboost 模型的结果来观察一些特征。

可以看到，伴随着模型结构的复杂，正确率反而下降了，这促使我们认识到一个非常重要的事情——模型并非越复杂越好，而是要适应模型的复杂度。注意，此处并非断定 CNN 一定比 XGboost 和前馈神经网络表现差，而是说在小任务付出相同的调试成本下，CNN 由于高 “Capacity” 过拟合问题仍然严重存在。

因此，我们下面的主要工作可以归结为 **如何选择适当的超参数和训练策略、简化神经网络结构，提高模型的泛化性，使得模型的 “capacity” 接近比赛数据的要求，使得深度学习模型取得比较好的效果**。

[4]High score：https://www.kaggle.com/code/twinpilgrim/spaceshiptitanic-xgboost-score-81-5

[5]Slides about capacity：https://cedar.buffalo.edu/~srihari/CSE676/5.2%20MLBasics-Capacity.pdf

由该比赛的高分模型均不约而同的规避了深度学习模型,可以说这并不是一件容易可以说是相当困难的事情,我们在进行了数百次的试错后,终于将结果提升到 5/2386(Top0.2%),以下是我们的总结和观察。

5.3 模型结构与超参数

我们最终选择了四层的前馈神经网络，其在几千个样本点和十几个特征的二分类任务上有充足的表达能力，更复杂的模型很难避免过拟合，关键代码如下。

```
class NNModel(nn.Module):
def __init__(self, input_size, drop_out_pct,gate):
    super(NNModel, self).__init__()
    self.gate = gate
    self.activation_func = torch.nn.ReLU()
    self.drop_out_pct = drop_out_pct

    self.fc1 = nn.Linear(input_size, 2048)
    self.dropout1 = nn.Dropout(drop_out_pct)
    self.batch_norm1 = nn.BatchNorm1d(2048)

    self.fc2 = nn.Linear(2048, 512)
    self.dropout2 = nn.Dropout(drop_out_pct)
    self.batch_norm2 = nn.BatchNorm1d(512)

    self.fc3 = nn.Linear(512, 16)
    self.dropout3 = nn.Dropout(drop_out_pct)
    self.batch_norm3 = nn.BatchNorm1d(16)

    self.fc4 = nn.Linear(16, 1)

    # Weight decay (L2 regularization) can be applied in the optimizer

def forward(self, x):
    x = self.activation_func(self.fc1(torch.tensor(x)))
    x = self.batch_norm1(self.dropout1(x))

    x = self.activation_func(self.fc2(x))
    x = self.batch_norm2(self.dropout2(x))

    x = self.activation_func(self.fc3(x))
    x = self.batch_norm3(self.dropout3(x))

    x = torch.sigmoid(self.fc4(x))

    return x
def predict_step(self, x):
    out=self.forward(x)
    label=out>self.gate
    label=label==1
    return label
```

我们下面介绍一些非常重要的超参数和训练策略，我们大致把它分为模型超参数和训练超参数两部分，并进行了大量的实验调研了他们的影响。

模型超参数有如下几个。

1. 激活函数

2. 正则化方式 $batch_norm$

3. Dropout

训练超参数有如下几个

1. 输出层的阈值，即代码中 $label = out > self.gate$ 的 self.gate 部分

2. 学习率

3. Batchsize

4. 训练次数和 earlystop

5.4 需要注意的点

在比赛中，可能一个样本的正负都会剧烈的影响名次，因此一定注意要在程序中提前固定随机数种子 seed 以便于稳定复现。

5.5 模型架构考虑和超参数选择

.. ....