lab5: 检测社交媒体虚假账户

PB21000023 廖墨峥

2024 年 2 月 1 日

目录

1 实验要求和目标

预测 test.json 中的 label 为”human” / ”bot”, 考察完整进行实验的能力, 涵盖任务规划至执行各阶段, 不仅限于模型构建.

2 实验原理

首先对数据进行预处理, 接着按照 4:1 的比例划分训练集和测试集, 采用**交叉验证法**, 对模型进行训练和测试, 最后取 5 次测试的平均值作为最终结果.

根据测试的结果, 选择最合适的模型, 并对rawtest.json中的数据进行预测, 输出到test.json中

3 实验步骤

3.1 导入数据

读取train.json中的数据, 并将其转换为pandas.DataFrame格式, 方便后续处理.

统计样本数量, 其中训练集有 1986 个样本和 44 个特征, 测试集有 150 个样本和 44 个特征, 测试集中label特征为空, 需要预测.

3.2 数据预处理

3.2.1 预处理准备工作

我们把数据预处理需要用到的函数放在一起, 打包为一个preprocess类, 方便后续调用. 下面对这个类的成员函数做出说明:

- parse_date(): 解析日期, 分别保存为年、月、日等特征
- str_similarit (): 计算两个字符串之间的余弦相似度
- extract_tld(): 根据**正则表达式提取域名后缀**
- label_encode(): 对 label 列进行编码,'bot' 置 1, 否则置 0

3.2.2 删除低质量的样本和特征

我们认为样本的完整性表明了样本质量, 因此如果一个特征的缺失值超过 50%, 我们就删除该特征. 类似地, 如果一个样本的缺失值超过 50%, 则删除该样本.

此时检查数据发现, 我们删掉了time_zone和utc_offset这两个特征, 简单分析这两个特征, 它们表示时区, 也就是地理位置信息, 考虑到全球各地都可能有 bot 账号, 因此删掉这两个特征问题不大.

3.2.3 特征分析与类型转换

首先合并重复列, 如果有列名相同, 只保留第一列.

接下来对各个特征进行处理,

create_at, 我们使用上面的函数解析日期, 把它转换为数值类型.

接下来有四个特征:id ,id_str ,name,screen_name, 我们认为**这四个特征起到的作用可以用两个特征来描述**, 而删去其它的特征:

- id: 用户的唯一标识符, 用于区分不同的用户
- name_corr: 用户name和screen_name的相似度, 用于区分虚假账户和真实账户

这是因为, 我们需要一个特征标识每一个用户, 而姓名和昵称的相似度可能与账号的身份有关, 例如一个 bot 可能懒得设置两个 name 于是它的name和screen_name就是一样的了. 而真人可能会因为顾及隐私而设置不同的名称.

location, 做独热编码

description, 在这里我们把它删去, 在**附录**中有详细的分析.

url, 我们认为域名本身没有什么意义, 因为 bot 也可能会使用各种与人类似的域名, 但是顶级域具有重要意义, 这是因为考虑到 bot 不太可能拥有.gov 这样的后缀, 因此我们决定**提取后缀名**代替url, 然后做独热编码

entities , 这个特征比较复杂, 我们只考虑它是否为空, 若为空则置 1, 否则置 0

接下来处理bool类型的特征, 我们把它转换为 0 和 1.

对于含有color类型的特征, 我们直接删掉, 颜色与账号身份并无太大关系.

lang, 独热编码.

translator_type, 独热编码.

label, 我们使用上面的函数对其进行编码,0 表示真实账户,1 表示虚假账户.

最后, 我们把所有的特征转换为数值类型, 并把缺失值填充为 0.

3.2.4 数据归一化

归一化前, 我们首先检查是否有整列值都相同的列, 如果有, 则删除该列, 因为它们的标准差为 0, 会产生除 0 错误.

然后我们使用MinMaxScaler函数对数据进行**归一化**, 把数据归一化到 01 之间.

3.2.5 特征选择与数据可视化

首先分开特征和标签, 观察到正例有 1130 个, 负例有 856 个, 两者数量相差不大.

考虑到数据集不是很大, 我们决定保留所有的特征.

接下来我们使用 t-SNE 进行降维, 并**可视化**数据集.

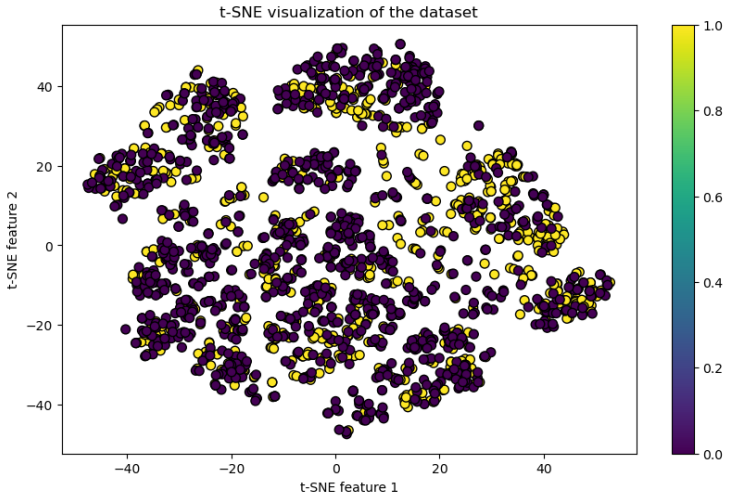

图 1: t-SNE 可视化

从图中可以看出, 数据集中的正负样本并没有明显的分离, 这说明我们的数据在高维空间才能分开.

3.2.6 数据集划分

我们使用KFold函数对数据集进行划分, 划分为 5 份, 其中 4 份作为训练集,1 份作为验证集.

3.2.7 对于测试集

对于测试集, 我们使用上面的函数对其进行预处理, 对于仅存在与训练集中的特征, 我们在测试集中添加一列, 并把其值置为 0; 对于仅存在与测试集中的特征, 我们直接删除该列.

3.3 模型建构与训练

每次选择 4 份数据进行训练,1 份数据用于测试, 最后取平均值, 输出格式为:$acc \pm \sigma$, 然后我们画出 **ROC 曲线**.

3.3.1 Logistic Regression

我们使用LogisticRegression函数构建模型, 设置最大迭代次数为 1000 次, 正则化系数为 1.2.

最终输出 $acc = 0.712 \pm 0.017$,ROC 曲线如下图所示:

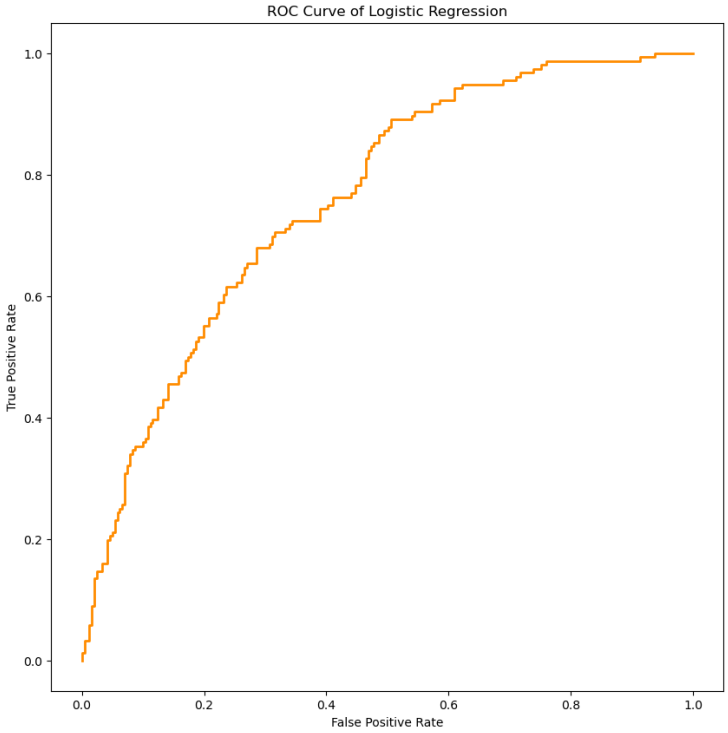

图 2: Logistic Regression

3.3.2 Decision Tree

我们使用 DecisionTreeClassifier 函数构建模型, 设置最大深度为 3.

最终输出 $acc = 0.729 \pm 0.018$,ROC 曲线如下图所示:

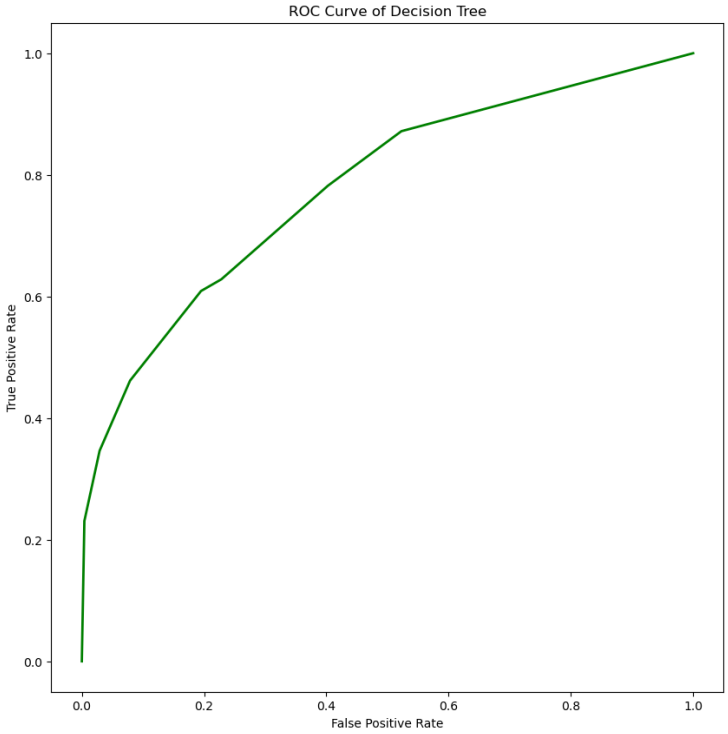

图 3: Decision Tree

3.3.3 Random Forest

我们使用RandomForestClassifier函数构建模型.
最终输出 $acc = 0.780 \pm 0.029$,ROC 曲线如下图所示:

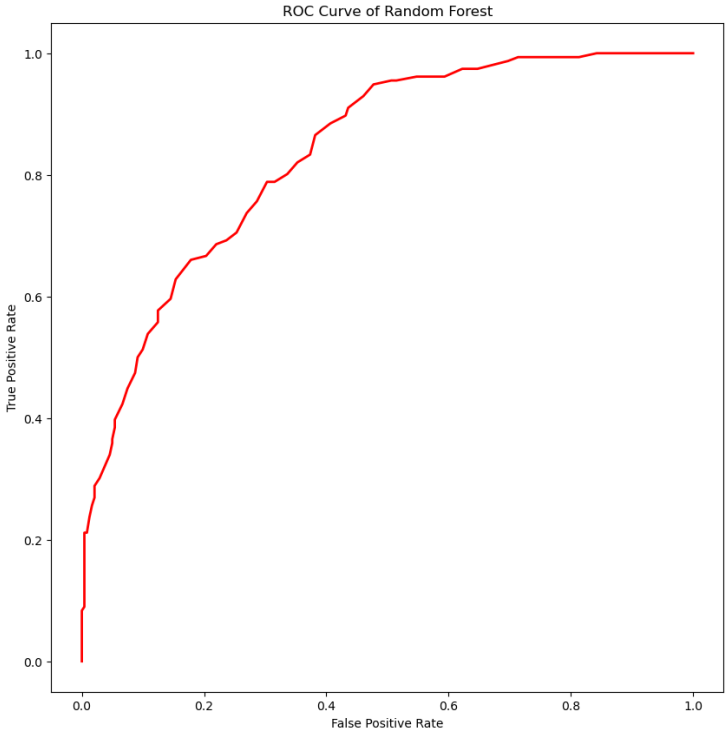

图 4: Random Forest

3.3.4 SVM

我们使用SVC函数构建模型, 设置线性核, 正则化系数为 1.2.
最终输出 $acc = 0.717 \pm 0.013$,ROC 曲线如下图所示:

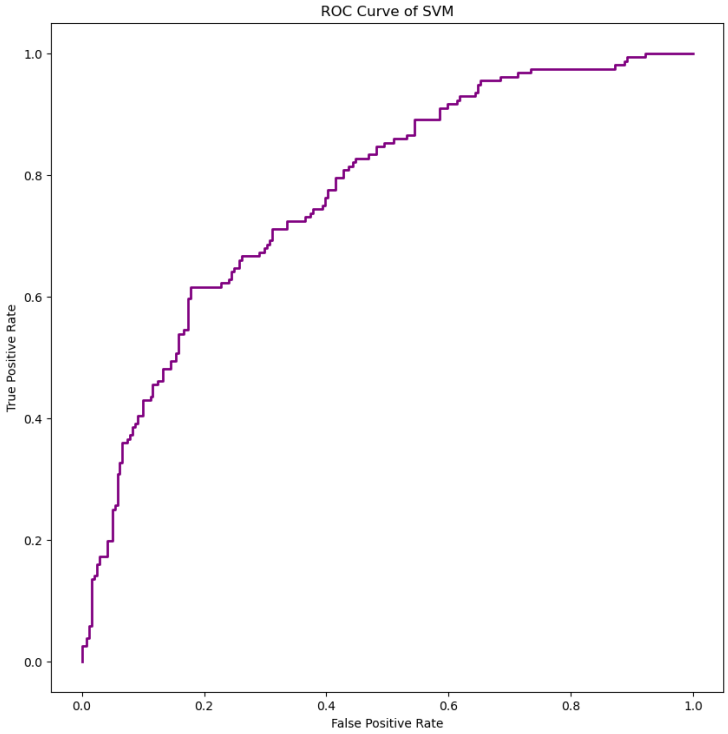

图 5: SVM

3.3.5 AdaBoost

我们使用AdaBoostClassifier函数构建模型.

最终输出 $acc = 0.760 \pm 0.0159$,ROC 曲线如下图所示:

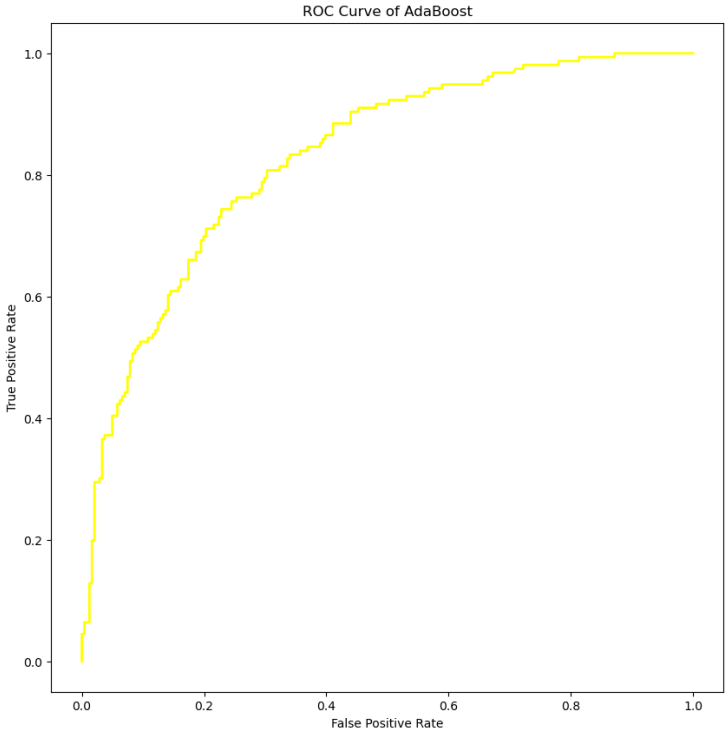

图 6: AdaBoost

3.3.6 GBDT

我们使用GradientBoostingClassifier函数构建模型.
最终输出 $acc = 0.768 \pm 0.021$,ROC 曲线如下图所示:

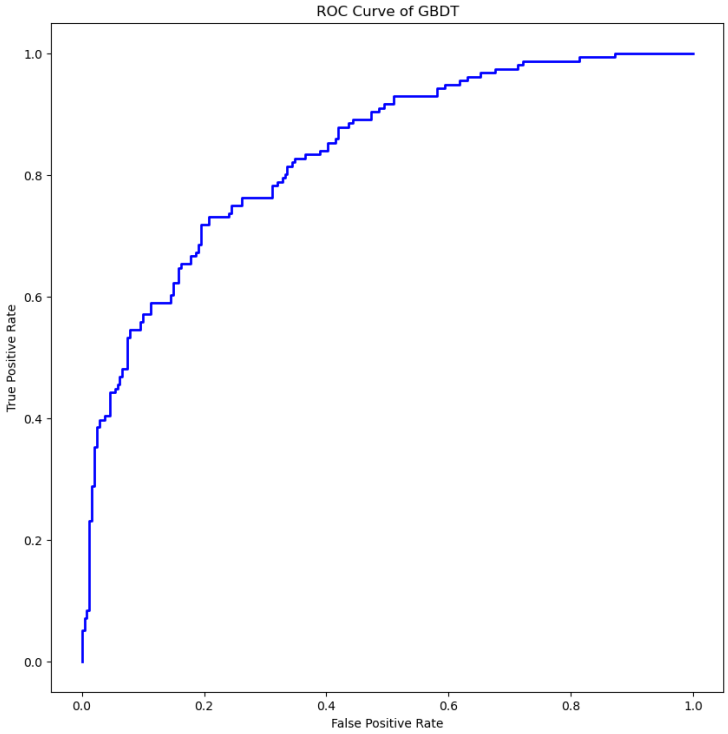

图 7: GBDT

3.3.7 MLP

我们使用MLPClassifier函数构建模型, 设置三层神经网络, 隐藏层神经元数量为 10, 激活函数为 Relu, 最大迭代次数为 1000, 正则化系数为 1.

最终输出 $acc = 0.718 \pm 0.020$,ROC 曲线如下图所示:

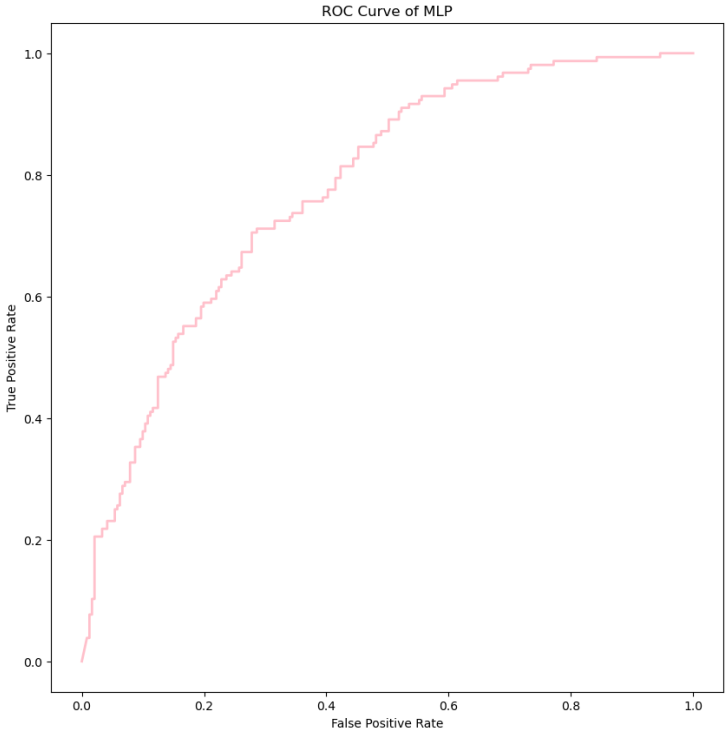

图 8: MLP

3.4 构建最终的分类器

我们最终选择RandomForestClassifier作为我们的分类器, 因为它的 *acc* 最高.

我们把所有的训练数据输入给它.

我们打印出特征的重要性图 (取前 20 个特征), 如下图所示:

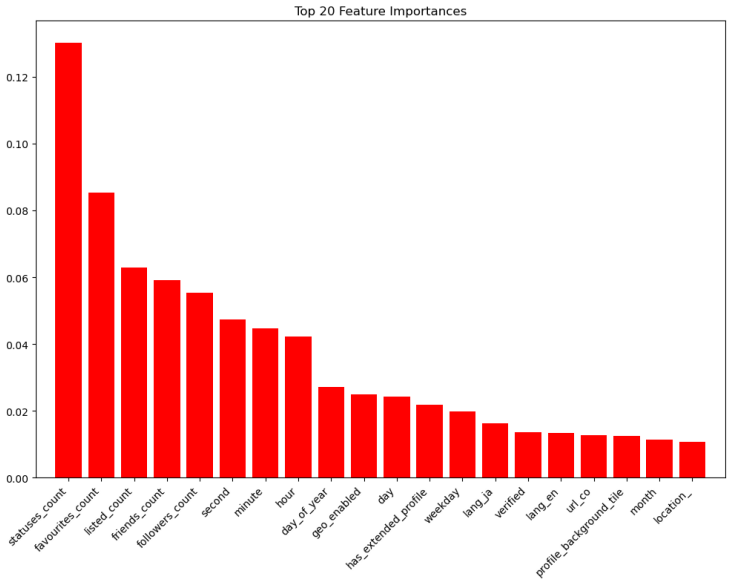

图 9: 特征重要性

接下来我们导入测试集, 对其进行预处理, 然后使用上面的分类器进行预测, 最后把结果写入test .json中.

4 实验结果

我们的最终结果为:

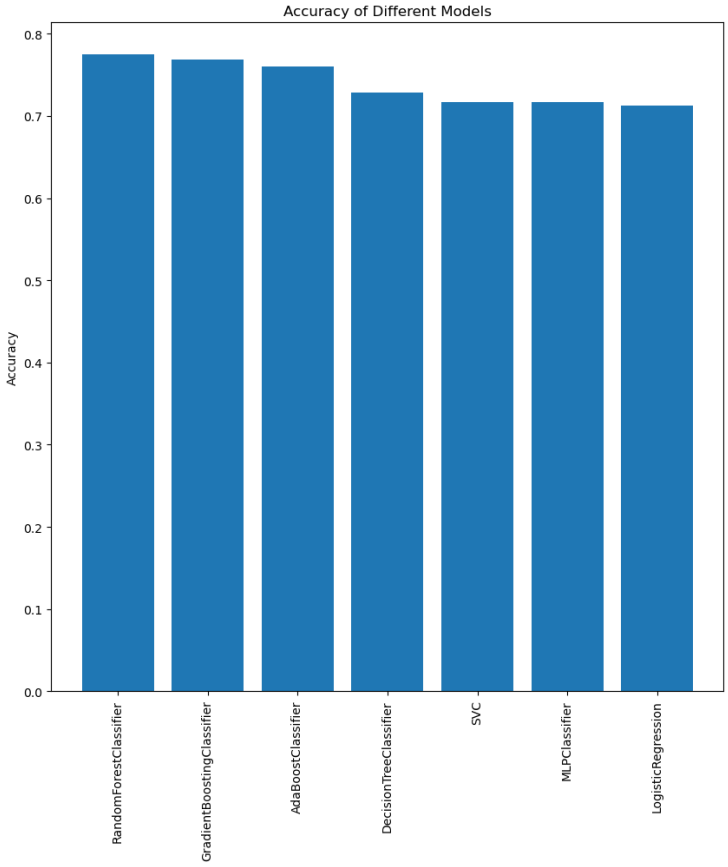

图 10: 实验结果

因此我们最终选择RandomForestClassifier作为我们的分类器, 因为它的 *acc* 最高.

5 实验分析

我们的数据集不是很大, 因此一些简单的模型也能取得不错的效果, 例如决策树, 但是随机森林的效果更好, 这是因为随机森林能够减少过拟合.

从这个角度考虑,MLP 没有取得很好的效果是很合理的. 同时, 在调参的过程中我发现**正则化系数对结果影响很大**, 这是因为我们的数据集不是很大. 在一开始我考虑过使用**数据增强**的方法, 但是考虑到数据集本身的特点, 例如id, 它是一个唯一标识符, 因此我们不能对它进行修改, 以及数据集中

还有很多特征被我做了独热编码, 因此数据增强的方法并不适用于这个数据集.

在缺乏数据的情况下, 集成学习方法普遍比单一模型效果好, 这是因为集成学习方法能够减少过拟合, 提高模型的泛化能力.

在先前我们观察特征图时看出, 数据集中的正负样本并没有明显的分离, 这说明我们的数据在高维空间才能分开, 因此类线性模型的效果也不是很好.

其实在选择保留大部分特征之前, 我尝试着计算过各个特征对 label 列的**相关系数与信息增益**, 结果发现, 大部分特征与 label 列的相关性都不是很高, 最终为了保证模型的泛化能力, 我选择保留了大部分特征. 我认为这可能也是模型的准确率很难突破 80% 的原因.

另外, 我们训练的模型预测结果的**方差都比较小**, 这说明数据集经过预处理之后所呈现的数据分布比较稳定, 这也是我们的模型能够取得不错的效果的原因.

6 亮点

- 对数据进行了详细的分析, 并对数据进行了预处理
- 深入分析了 description 特征, 使用了 distilbert 模型对其进行了处理
- 对数据特征进行了可视化, 并对数据进行了降维
- 使用了交叉验证法, 提高了模型的泛化能力
- 尝试了多种模型, 并对比了它们的效果
- 画出了 ROC 曲线, 并比较了它们的 AUC
- 分析了各个模型的优缺点, 并选择了最合适的模型
- 对实验结果进行了分析, 并给出了原因

A 对特征 description 的处理

我们首先将description中的缺失值填充为unknown, 然后把description和label这两列单独拿出来.

接下来, 调用预训练的 distilbert −base−uncased模型帮助我们处理这个文本特征.

接着, 对训练集进行分词处理, 并把数据打包为torch. utils .data.DataLoader格式, 设置batch_size=16方便后续处理.

然后, 我们使用 distilbert −base−uncased模型在训练集上进行微调, 设置学习率为 $5e-5$,$epoch=30$.

最后, 我们使用微调过的模型在验证集上预测, 得到的结果如下图所示:

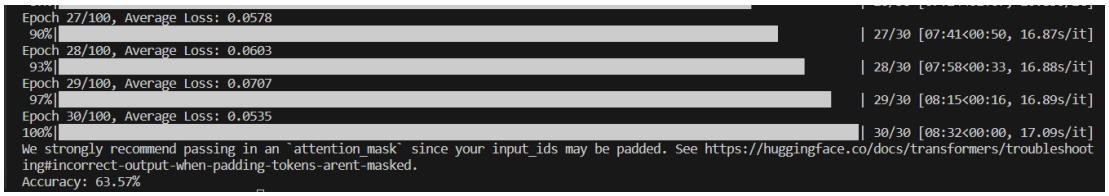

图 11: bert

预测正确率只有 0.63 左右, 这说明description这个特征对于我们的模型来说并没有太大的意义, 因此我们决定删去这个特征.

下面附有description的预处理代码,python 版本为 3.9(jupternotebook 的 python 版本为 3.10, **请注意区分**)

B 代码

```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizerFast, DistilBertForSequenceClas
import torch
from torch.optim import AdamW
from tqdm import tqdm


def load_data(path):
    df = pd.read_json(path)
    df_expanded = df['user'].apply(lambda x: pd.Series(x))
    df = pd.concat([df.drop('user', axis=1), df_expanded], axis=1)
```

```
    return df

class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.i
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

class preprocess:
    @staticmethod
    def label_encode(strr):
        """
        对指定列进行标签编码
        """
        if strr == 'bot':
            return 1
        else:
            return 0

if __name__ == '__main__':
    data_train = load_data("./data/train.json")
    data_test = load_data("./data/test.json")
    data_train.shape, data_test.shape

    data_train['label'] = data_train['label'].apply(lambda x: preprocess.lab
```

```
data_train['description'] = data_train['description'].fillna('unknown')

data_description = data_train[['description', 'label']]
data_description['label'] = data_description['label'].astype(int)

train_texts, val_texts, train_labels, val_labels = train_test_split(data
tokenizer = DistilBertTokenizerFast.from_pretrained('./distilbert-base-u
model = DistilBertForSequenceClassification.from_pretrained('./distilber

# 分词处理
train_encodings = tokenizer(train_texts.tolist(), truncation=True, paddi
val_encodings = tokenizer(val_texts.tolist(), truncation=True, padding=T

train_dataset = Dataset(train_encodings, train_labels.tolist())
val_dataset = Dataset(val_encodings, val_labels.tolist())

# 创建数据加载器
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=16,
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=16, shu

device = torch.device('cuda') if torch.cuda.is_available() else torch.de
model.to(device)
model.train()

optim = AdamW(model.parameters(), lr=5e-5)

for epoch in tqdm(range(30)):
    total_loss = 0
    model.train()

    for batch in train_loader:
        optim.zero_grad()
```

```
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids, attention_mask=attention_mask, labels
            loss = outputs.loss
            total_loss += loss.item()

            loss.backward()
            optim.step()

        # 计算平均损失
        avg_loss = total_loss / len(train_loader)
        print(f"Epoch {epoch + 1}/{100}, Average Loss: {avg_loss:.4f}")

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in val_loader:
            inputs = batch['input_ids'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(inputs)

            _, predicted = torch.max(outputs.logits, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = correct / total
    print(f'Accuracy: {accuracy * 100:.2f}%')
```