银行客户流失预测实验报告

杨坤泽 2023210799

October 2023

1 数据处理

先分析实验数据，对数据进行预处理，如提取特征和标签、舍弃未知数据 (数据中的 Unknown 项)，以及对特征的文字数据进行编码。其中的编码方式应用 sklearn 库中的 LabelEncoder 实现，之后对所有数据进行归一化和标准化。

观察实验数据，发现标签中的 0 和 1 的数量并不均衡 (1 类样本数量多于 0 类样本)，因此考虑采用集成方法，利用多个对 1 类欠采样和所有 0 类构成的样本训练得到的子模型求期望得到最终的训练模型。在代码实现中采用随机采样的方法，在每个 loop 循环中对 1 类完成欠采样并划分总数据集，划分方式为 80% 的训练集与 20% 的测试集。具体实现代码如下：

```
    for num in range(loop):
        #split train and test data
        idx_zero = np.where(y == 0)[0]
        idx_one = np.array([i for i  in range(x.shape[0]) if i not in
                                            idx_zero])
        np.random.shuffle(idx_one)
        idx_one = idx_one[:idx_zero.shape[0]]
        x_split_zero, x_split_one, y_split_zero, y_split_one = x[idx_zero],
                                            x[idx_one], y[idx_zero], y[
                                            idx_one]
        train_zero_x, test_zero_x, train_zero_y, test_zero_y=
                                            train_test_split(x_split_zero,
                                             y_split_zero, test_size=0.2)
        train_one_x, test_one_x, train_one_y, test_one_y= train_test_split(
                                            x_split_one, y_split_one,
                                            test_size=0.2)
        train_x, test_x, train_y, test_y = np.concatenate((train_zero_x,
                                            train_one_x), axis=0), np.
                                            concatenate((test_zero_x,
                                            test_one_x), axis=0), np.
                                            concatenate((train_zero_y,
                                            train_one_y), axis=0), np.
                                            concatenate((test_zero_y,
                                            test_one_y), axis=0)
        train_y ,test_y = np.expand_dims(train_y, axis=1), np.expand_dims(
                                            test_y, axis=1)
```

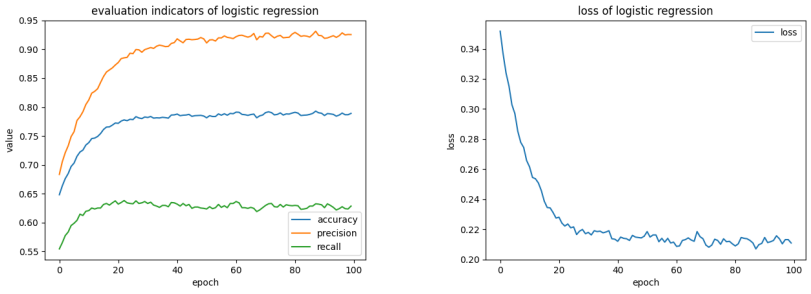
2 基于逻辑回归的算法实现

对于逻辑回归方法，利用 numpy 库自定义 logistic_regression 类实现。输入包括训练数据 x、标签 y、学习率 lr、迭代数 iter、方便打印的子模型序号 i_th 以及正则化系数 lambda。对于梯度下降方法，分别手写得到 SGD_gradient 与 BGD_gradient 方法如下：

```
#BGD method(sample_rate = 1/5)
def BGD_gradient(self):
    idx = np.random.randint(0, self.x.shape[0], self.sample_num)
    gradient = np.dot(np.transpose(self.x[idx, :]), (self.sigmoid(np.dot
                                        (self.x[idx, :], self.w))-self
                                        .y[idx].astype('float')))/self
                                        .sample_num + self.lamda*self.
                                        w.astype('float')
    return gradient

#SGD method
def SGD_gradient(self):
    idx = np.random.randint(0, self.x.shape[0])
    gradient = np.transpose(self.x[idx, :])*(self.sigmoid(np.dot(self.x[
                                        idx, :], self.w))-self.y[idx].
                                        astype('float')) + self.lamda*
                                        self.w.astype('float')
    return gradient
```

并定义 accuracy、precision、recall 等函数方便计算指标。最终经过 100 轮训练得到的指标曲线和误差曲线如下：



(a) 指标曲线　　(b) 误差曲线

注意到模型收敛较快，实现了近 80% 的准确率、93% 的精确率和 65%

的召回率。其中召回率较低，说明模型对正类的分类正确率较低，而较高的精确率说明预测正类正确的样本比例很高，这也与训练数据的正负类样本数量有关。将各个子模型的训练结果集成，最终得到的模型结果如下：

```
test accuracy of all models of log_reg:  0.8146067415730337
test precision of all models of log_reg:  0.9666666666666667
test recall of all models of log_reg:  0.651685393258427
```

可以注意到相较于子模型，集成得到的模型具有更优的指标结果，说明集成方法能够有效改善样本不均衡对模型的影响。

3 基于回归的分类算法实现

对于回归方法，利用 torch 库定义 linear_regression 类实现。各项参数包括训练数据 x、标签 y、学习率 lr 等。线性模型的前向计算函数定义如下：

```
    def forward(self, x):
        return torch.matmul(self.w, torch.transpose(x, 0, 1)) + self.bias
```

并定义 accuracy、precision、recall 等函数方便计算指标，过程中注意将张量转化为 numpy 数组方便计算。其中在训练部分利用 pytorch 中的 nn.MSELoss() 自定义均方误差函数，优化器采用 SGD 优化器，通过 TensorDataset 综合训练数据和标签，并利用 DataLoader 进行训练。在训练过程中注意将梯度清 0 后再反向传播，避免梯度的累积。关键代码如下：

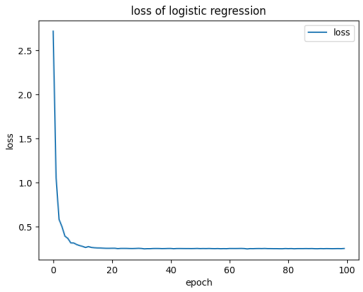
```
        model = linear_regression()
        mseloss = nn.MSELoss()
        opt = optim.SGD(model.parameters(), lr=0.005)
        dataset = TensorDataset(torch.from_numpy(mapping(train_x).astype('
                                                float32')), torch.from_numpy(
                                                train_y.astype('float32')))
        train_dl = DataLoader(dataset, batch_size=16, shuffle=True)
        tbar = tqdm(range(iter_num))
        i = iter_num
        for _ in tbar:
            tbar.set_description('Training ' + str(num) + '-th regression
                                                    sub-model')
            for xdl, ydl in train_dl:
                pred = model(xdl)
                loss = mseloss(pred, ydl)
```

```
            opt.zero_grad()
            loss.backward()
            opt.step()
```

代码中保留了利用 numpy 库手写的 regression 类以及训练曲线，最终训练效果并不理想，可能是由于梯度传播与计算的过程中存在问题。

最终经过 100 轮训练得到的指标曲线和误差曲线如下：

(c) 指标曲线



(d) 误差曲线

注意到模型收敛相较于逻辑回归模型更快，实现了近 65% 的准确率、精确率和召回率。与逻辑回归模型最终实现的效果不同，三者非常接近。对于线性回归模型而言，本身不可能实现对于分类问题的小误差逼近；对于最终预测的实现需要引入符号函数对结果进行硬判决，在硬判决的过程中可能会改善线性问题的影响。与逻辑回归问题的区别也在于此，逻辑回归采用 sigmoid 函数，通过非线性提高模型的拟合能力。在回归问题中，想要提高模型的拟合能力就需要引入非线性基函数实现，可以进一步提高模型的性能。

将各个子模型的训练结果集成，最终得到的模型结果如下：

```
test accuracy of all models of reg:  [0.86235955]
test precision of all models of reg:  [0.89570552]
test recall of all models of reg:  [0.82022472]
```

可以注意到相较于子模型，集成得到的模型指标更优。通过观察发现回归模型相较于逻辑回归模型训练结果不稳定，与子模型训练时欠采样的随机性有关。回归模型能够实现更高的召回率，但精确率就相对较低。通过不同的指标要求，可以采用不同的模型与方法实现目标。