课程名称：人工智能原理（2023 年秋）

授课教师：江瑞教授

人工智能原理第二次编程小作业

刘馨阳 2021012910 新雅智-1

1.逻辑回归得到字体粗细的预测二分类

使用 scikit-learn 库内的模块即可完成上述任务：

```
from sklearn.linear_model import LogisticRegression
```

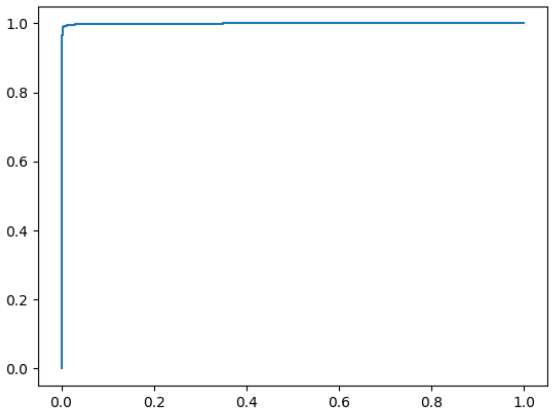
该库内同样有用于求解各个评价指标的函数，故引用即可：

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, roc_auc_score
```

最终通过模型训练得到参数，进行预测后结果如下：

```
accuracy score: 0.9929777379351561
precison score: 0.9946808510638298
recall_score: 0.9914580265095729
f1_score: 0.9930668240153414
auROC: 0.9989261131171567
```

ROC 曲线如下图：



从上述数据指标和图像指标来看，模型训练的效果比较不错，各个决策阈值下的性能都比较完备（各个数据 score 值接近 1），在较高特异度情况下也能获得较高敏感度（ROC 曲线面积接近 1）。

2.softmax 回归得到数字大小的预测十分类

使用第一题中的模块内置函数同样可以完成本任务，此处不再赘述。
值得提及的是，由于多分类和单分类最终的结果种类数目不同，在求解评价指标的值时会出

现一些区别。

a.由于分类的数目进行了扩展，在计算召回率、精确率和 F1-score 时，不能再采取之前的简单的除法计算，需要在样例层面进行平均，因此在这里需要添加参数 macro:

```
print("macro-precison score:", precision_score(test_number,
test_number_pred, average = "macro"))
print("macro-recall_score:", recall_score(test_number,
test_number_pred, average = "macro"))
print("macro-f1_score:", f1_score(test_number, test_number_pred,
average = "macro"))
```

b.由于分类的数目进行了扩展，在计算 auROC 时，需要使用独热码作为模块中函数的实参:

```
test_number_one_hot = np.zeros((6693, 10), dtype = int)
for i in range(6693):
    test_number_one_hot[i][test_number[i]] = 1
print("auROC:", roc_auc_score(test_number_one_hot, test_number_probas,
average = "micro"))
```

c.对于混淆矩阵，同样直接从库中引用函数即可:

```
from sklearn.metrics import confusion_matrix
```

最终通过模型训练得到参数，进行预测后结果如下:

```
accuracy score: 0.8579112505602868
macro-precison score: 0.8559698632753072
macro-recall_score: 0.8563822233367542
macro-f1_score: 0.8559100203131706
auROC: 0.9812885895870433
[[589   0   5   2   2  13  10   5   4   0]
 [  1 710   4   6   1   6   3   1  15   1]
 [ 17  19 559  23  12   6  12  12  24   7]
 [  7   5  25 580   4  38   1  15  19   5]
 [  4   1   5   2 581   3  14   5   8  49]
 [ 19   8   7  37  13 452  11  18  27   4]
 [ 16   2  11   2  12  15 572   2   6   1]
 [  5   4  21   8   8   2   0 623   3  24]
 [  9   9  19  27  11  26  10  14 523  12]
 [  9   3   4   7  39   7   3  27   8 553]]
```

从上述数据指标和混淆矩阵的具体数值来看，模型训练的效果还能接受，各个决策阈值下的性能勉强完备（各个数据 score 值比较接近 1）。

3.前馈神经网络得到数字大小的预测十分类

在固定的 epoch 次数内进行迭代，batch_size = 512，由计算[39980/512] = 78，得知每一次 epoch 会进行 79 次迭代。

在每一次迭代中，先根据已有的参数进行一次前向传播，通过给定的交叉熵求解损失，然后再进行反向传播求解梯度并且更新参数：

```
train_num_pred = model(picture)
loss = criterion(train_num_pred, train_num)
loss.backward()
optimizer.step()
```

另外，通过累计每一次迭代的损失求和，对于迭代次数做平均得到一个 epoch 的平均损失，可以以此监控模型的训练情况。

```
runningloss += loss.item()
        if batch_times % 79 == 78:
            print(epoch+1, runningloss/79)
            losses_list.append(runningloss/79)
            runningloss = 0.0
```

个人补充的部分代码如下：

```
for epoch in range(num_epochs):

    runningloss = 0.0
    for batch_times, data in enumerate(train_loader, 0):
        picture, train_num = data

        optimizer.zero_grad()

        # print(batch_times)

        train_num_pred = model(picture)
        loss = criterion(train_num_pred, train_num)
        loss.backward()
        optimizer.step()

        runningloss += loss.item()
        if batch_times % 79 == 78:
            print(epoch+1, runningloss/79)
            losses_list.append(runningloss/79)
            runningloss = 0.0

print(losses_list)
```

在进行评价指标的求解时，各个函数基本按照第二题的方式进行，因此不再赘述。

根据多次实验修改学习率和 epoch，最终在 lr = 0.001，num_epoch = 500 时有预测结果如下：

```
accuracy score: 0.949350067234424
macro-precison score: 0.9492810050764797
```

```
macro-recall_score: 0.9493021382323408
macro-f1_score: 0.9489687942565018
auROC: 0.9821484383658997
[[622   0   0   0   0   1   5   1   1   0]
 [  4 731   1   1   0   0   4   3   3   1]
 [ 16   3 643   8   3   0   4   5   8   1]
 [ 10   3   4 653   0  10   1   3   7   8]
 [  4   1   2   0 641   0   2   2   4  16]
 [  5   0   2   9   1 560   4   3  10   2]
 [  8   4   2   0   0   7 616   0   2   0]
 [  6   3   5   3   3   0   0 659   4  15]
 [ 16   1   2   3   3   9   2   4 615   5]
 [  8   3   1   4  13   5   0   8   4 614]]
```

从上述数据指标和混淆矩阵的具体数值来看，模型训练的效果比较好，各个决策阈值下的性能相对完备（各个数据 score 值比较接近 1），同时容易看出前馈神经网络的训练效果明显优于softmax回归（考虑softmax回归共1000个epoch，而前馈神经网络仅使用500个epoch）。

Plus：在训练过程中发现，损失函数存在某些局部最低点和数值高原的情况，而当前算法除了调整学习率之外，并没有很好的能够确保完全避免出现这些现象出现的方法（e.g. loss = 2.303）