代码和数据集链接：

<https://github.com/Ekeulseuji/An-Optimized-BP-Neural-Network-Model-Based-on-Genetic-Algorithm-for-Human-Pre-miRNA-Classification>

1. 数据特征（一共用了923个数据 来自miRBase数据库的）

原数据特征： | 名称 | 序列 | 类别 |

然后先根据序列的组成计算了每个序列的

| 点括号表达方式 | 自由能 | mature 5p' 序列 | mature 3p' 序列 |

一張含有 文字, 螢幕擷取畫面, 字型, 圖表 的圖片

自動產生的描述 左边是Pre-miRNA分子的形态（是发夹的样子）

——————————————————————————————————

然后整理出了25个新特征

（参考了这篇文章 https://www.nature.com/articles/s41598-018-25578-3#Sec2）：

# harpin\_fe 是 利用 ViennaRNA Package中的 RNAfold计算得到的 Harpin free energy 自由能（原样保留）

# harpin\_len 是 Pre-miRNA长度（就是计算序列中包含多少个字母或核苷酸）

# mature5p\_len 是 包含 mature 5p序列的单链RNA长度（就是计算序列中包含多少个字母或核苷酸）

# mature3p\_len 是 包含 mature 3p序列的单链RNA长度（就是计算序列中包含多少个字母或核苷酸）

# interarm\_len 是 arm5p和arm3p序列之间的单链RNA长度（就是计算harpin\_len - mature5p\_len - mature3p\_len）

# n\_short\_loop 是 长度小于4个核苷酸的环状结构的数量（计算dotbracket中 '.' '-' 和 '|' 的数量）

# n\_long\_loop 是 长度大于等于4个核苷酸的环状结构的数量（计算dotbracket中 '|||' '||'和 '|-|'的数量）

# overhang\_len 是 过挂长度 (harpin中茎部未配对核苷酸的数量) (就是计算 (mature5p\_len - mature3p\_len))

# terminal\_len 是 末端环长度 （计算dotbracket中第一个分隔符之前的长度）

# mature5p\_A 是 包含 mature 5p序列的单链RNA里面的 A content（就是计算序列中包含多少个字母A）

# mature3p\_A 是 包含 mature 3p序列的单链RNA里面的 A content（就是计算序列中包含多少个字母A）

# interarm\_A 是 arm5p和arm3p序列之间的单链RNA里面的 A content（就是计算序列中包含多少个字母A）

# harpin\_A 是 整个发夹结构里面的 A content（就是计算序列中包含多少个字母A）

# mature5p\_U 是 包含 mature 5p序列的单链RNA里面的 U content（就是计算序列中包含多少个字母U）

# mature3p\_U 是 包含 mature 3p序列的单链RNA里面的 U content（就是计算序列中包含多少个字母U）

# interarm\_U 是 arm5p和arm3p序列之间的单链RNA里面的 U content（就是计算序列中包含多少个字母U）

# harpin\_U 是 整个发夹结构里面的 U content（就是计算序列中包含多少个字母U）

# mature5p\_G 是 包含 mature 5p序列的单链RNA里面的 G content（就是计算序列中包含多少个字母G）

# mature3p\_G 是 包含 mature 3p序列的单链RNA里面的 G content（就是计算序列中包含多少个字母G）

# interarm\_G 是 arm5p和arm3p序列之间的单链RNA里面的 G content（就是计算序列中包含多少个字母G）

# harpin\_G 是 整个发夹结构里面的 G content（就是计算序列中包含多少个字母G）

# mature5p\_C 是 包含 mature 5p序列的单链RNA里面的 C content（就是计算序列中包含多少个字母C）

# mature3p\_C 是 包含 mature 3p序列的单链RNA里面的 C content（就是计算序列中包含多少个字母C）

# interarm\_C 是 arm5p和arm3p序列之间的单链RNA里面的 C content（就是计算序列中包含多少个字母C）

# harpin\_C 是 整个发夹结构里面的 C content（就是计算序列中包含多少个字母C）

把数据集分成6训练:2验证:2测试集 做特征选择（不知道是不是这样做的）

分别用了线性回归、Lasso逻辑回归、Ridge逻辑回归、Ridge逻辑回归前向特征选择、Ridge逻辑回归反向特征选择、还有随机森林特征重要度排序 做了特征选择

然后用 不同方法选出来的特征 拟合SVM基线模型

最后选一组在验证集上表现最好的特征（准确率最高）用在后续的训练里面

2. 模型思路

整体思路：用遗传算法给BP神经网络搜出最佳的权重和偏置，帮助最小化神经网络的损失

遗传算法思路：

目标是优化参数，找到最好的参数；步骤是先随机产生一些个体的基因型（种群）（11010110011101），每一条个体都是有相对应的表现型（实数）的，这些个体都是优化问题的可能解；然后对种群进行遗传过程，先选择个体作为上一代，适应度高的（在解决问题中比较优的）个体会更有可能被选择到，随机交叉产生新的下一代，新的下一代根据设定的概率可能会在基因型的随机位置发生变异，增加物种多样性。最后满足适应度阈值或者满足最大迭代次数了就不再对种群做遗传了，直接输出适应度最高的个体作为最优解（最佳的权重和偏置）

3. 模型建立步骤（从第38个单元格开始是GABP）

1. 用pytorch建立BP神经网络模型，初始权重设置成遗传算法得到的最优解

input\_layer = torch.nn.Linear(in\_features=13, out\_features=5)

relu\_layer1 = torch.nn.ReLU()

dense\_layer = torch.nn.Linear(in\_features=5, out\_features=3)

relu\_layer2 = torch.nn.ReLU()

output\_layer = torch.nn.Linear(in\_features=3, out\_features=1)

sigmoid\_layer = torch.nn.Sigmoid()

# 把遗传算法得到的最优解（权重向量）转换为神经网络模型的权重字典形式

best\_solution\_weights = torchga.model\_weights\_as\_dict(model=model, weights\_vector=solution)

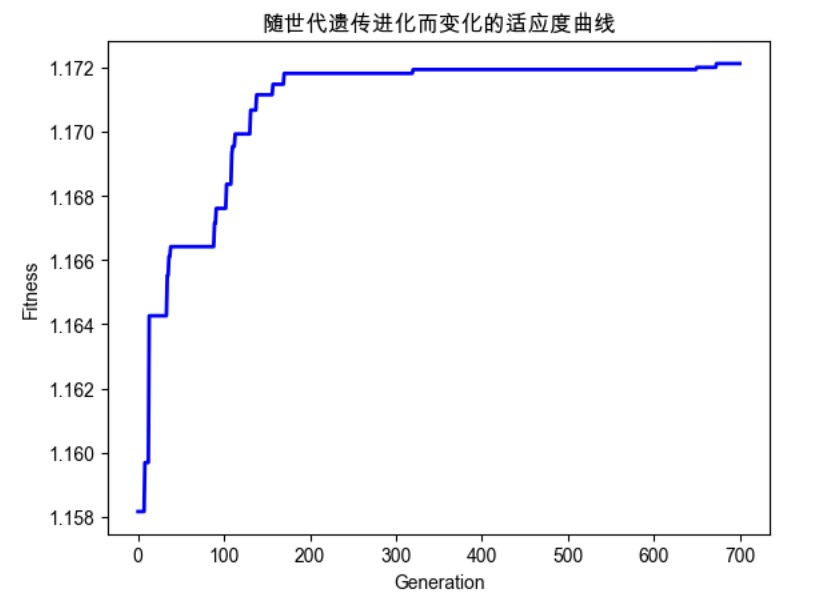
# 更新模型的权重

model.load\_state\_dict(best\_solution\_weights)

2. 再用pygad包选定参数：进化600代，轮盘赌，单点交叉，随机变异（17%）等

适应度函数用了这个 每个世代最佳个体的适应度（降低损失的表现）会被记录下来最后画图

solution\_fitness = 1.0 / (loss\_function(predictions, data\_outputs).detach().numpy() + 0.00000001)



3. 每次/每个世代选定好参数之后在神经网络里用验证集算训练集+验证集的损失和准确度，在训练完成之后画损失和准确率随世代变化的折线图（这样做对吗）

# 当前最佳权重

best\_solution\_weights = torchga.model\_weights\_as\_dict(model=model, weights\_vector=ga\_instance.best\_solution()[0])

# 用到神经网络里 用验证集算损失和准确率

model.load\_state\_dict(best\_solution\_weights)

# 在每一代结束时进行验证

val\_inputs = torch.from\_numpy(new\_X\_val.values).float()

val\_outputs = torch.from\_numpy(y\_val.values).float()

val\_outputs= val\_outputs.reshape((val\_outputs.shape[0], 1))

# 进行预测 这里 val\_predictions输出的是概率 然后用BCEWithLogitsLoss()算了损失

val\_predictions = model(val\_inputs)

val\_loss = loss\_function(val\_predictions, val\_outputs).detach().numpy()

# 将概率值转换为包含0和1的预测目标列

val\_predicted\_labels = (val\_predictions >= threshold).int()

val\_true\_labels = val\_outputs

val\_accuracy = (val\_predicted\_labels == val\_true\_labels).sum().item() / len(val\_true\_labels)

一張含有 圖表, 文字, 螢幕擷取畫面, 行 的圖片

自動產生的描述

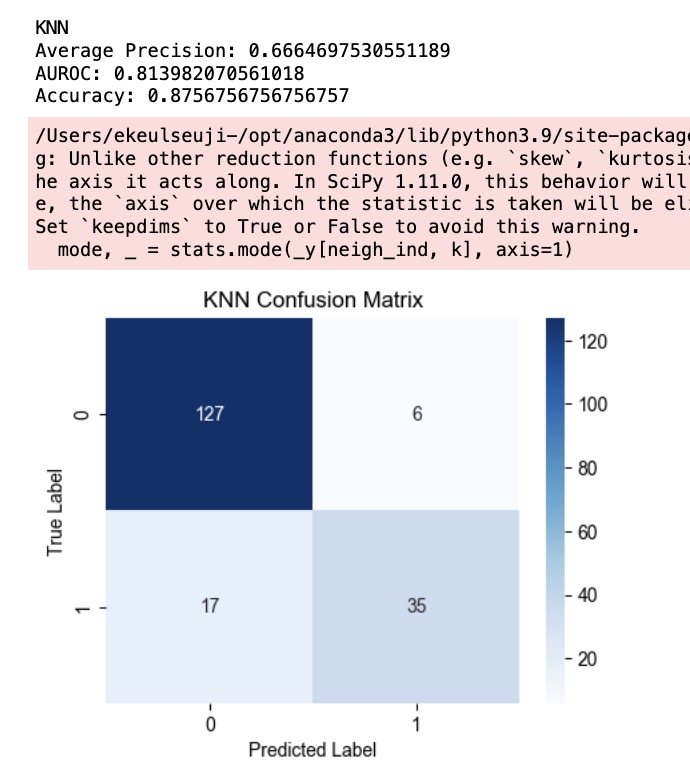
应该可以再改进一下 世代数设置多了？

4. 模型求解结果

最后用训练好的模型在测试集上做分类



对比其他模型的表现 平均精度和ROC下面积有明显提升 准确率91.8还可以

一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

自動產生的描述 一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

自動產生的描述 一張含有 文字, 螢幕擷取畫面, 圖表, 字型 的圖片

自動產生的描述

其他模型也还行，是不是数据集太小了

之前6:2:2不应该分验证集出去？特征选择也不应该用那么多方法去比较？可以直接做的（因为好像结果都差不多，但是参考了之前提到的那篇文章他们用了不同的方法去选择特征）

如果7:3再做一次，那么之前验证集在每个世代最后输出模型的损失和准确率就不能做出来了是不是啊