# 建模思路

2024.5.26

1. 总体思路

目的：根据不同考生分数、偏好等因素，为考生提供最优专业与学校选择建议。

衡量标准：各个选择（学校+专业）选择指数、效用值、效用概率、录取概率，使尽可能大。

目标函数：

#todo:完善录取概率计算

#为考生i高考分数排名，为该选择的招生人数，具体函数形式需要讨论

假设：

* 各个选择的各项客观指标不会变化随考生变化而变化，而各项主观指标则会随考生变化而变化
* 各个考生对各项指标的权重分配和不会随选择变化而变化
* 选择指数分为2个部分，效用概率和录取概率，效用概率满足MNL模型，录取概率满足？

1. 参数指标、设置与量化

说明：，大部分指标均需要根据其它参数进行折算

#todo每个指标的具体折算方法

* 1. 客观指标：

学校相关：

：学校排名指数（参考QS、泰晤士、USnews等）

：学校综合实力指数（参考学校等级（如985、211、双一流等）、学校科研经费、师资力量）

：学校地理位置（参考城市等级、交通便利程度、与家乡距离等）

#可以考虑把与家乡距离单独拎出来

：学校环境与设施情况（参考图书馆、宿舍、体育馆、实验室等）

：学校校友资源（参考知名校友数量、？等）

……

专业相关：

：专业就业压力（参考招聘市场需求量、毕业生数量、平均薪水等）

：专业学习难度与压力（参考主修课程平均成绩、学长评价等）

：专业发展潜力（参考平均薪水、国家政策支持度、专业跳槽率、毕业生去向分布等）

：专业社会影响力（参考？）

……

* 1. 主观指标：

：专业的兴趣与爱好

：家庭经济压力（参考学费、生活成本、消费成本、收入情况等）

……

1. 模型
   1. 调查数据
      1. 面向往届考生

数据用于训练模型、测试模型

原始数据：考生的最终录取去向、考生对各项指标的偏好情况和、考生的主观指标。其中根据问卷设置，

* + 1. 面向应届考生

数据用于模型应用、测试模型

原始数据：考生的最终录取去向、考生对各项指标的偏好情况和、考生的主观指标。其中根据问卷设置，

* 1. 既有数据
     1. 客观指标数据

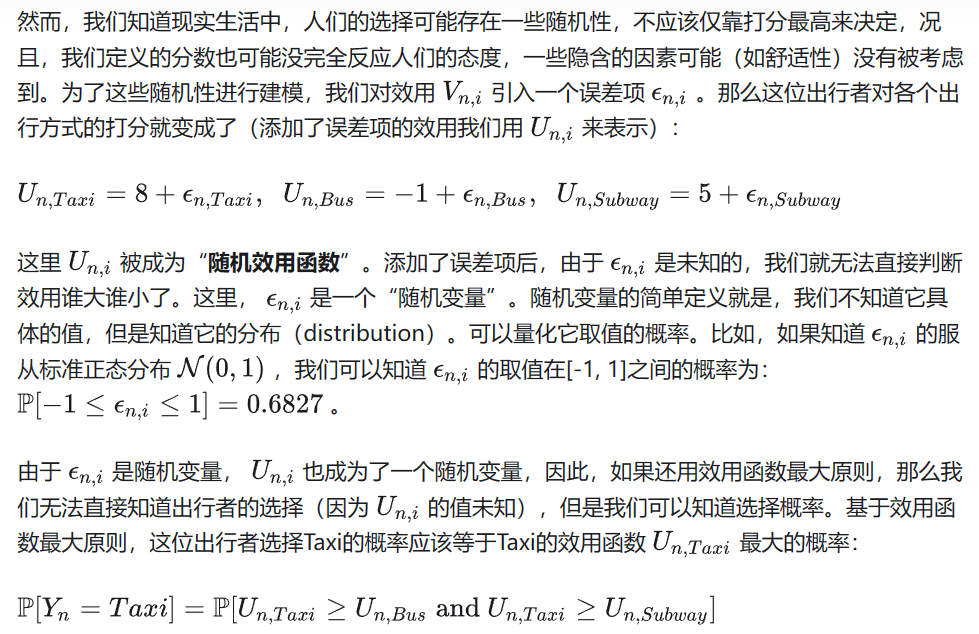
待收集

* + 1. 各学校专业的招生人数和最低录取分数线

为该选择的最低分数排名，为该选择的招生总人数

* + 1. 考生排名和一分一段表
  1. 数据处理
     1. 效用模型

这里引入一个随机效用函数：



* + 1. 录取模型

#todo:完善录取模型函数

1. 确定基本假设

录取概率与考生的高考分数排名和选择的学校专业招生人数有关

2. 设计录取概率函数

2.1 选择逻辑函数

选择一个适合的逻辑函数来表示录取概率，常用的是 Sigmoid 函数。

[ f(x) = \frac{1}{1 + e^{-x}} ]

2.2 建立录取概率函数

使用选定的逻辑函数，将考生的高考分数排名 ( Ri ) 和选择的学校专业招生人数 ( N{ij} ) 考虑进来，建立录取概率函数：

( P\_i )：考生 ( i ) 的录取概率。

( R\_i )：考生 ( i ) 的高考分数排名。

( N\_{ij} )：考生 ( i ) 选择的学校专业 ( j ) 的招生人数。

( a, b, c )：待定参数，用于调整函数的斜率和截距。

[ Pi = \frac{1}{1 + e^{-(aRi + bN\_{ij} + c)}} ]

其中 ( a, b, c ) 是待定参数，用于调整函数的斜率和截距

比方说，参数 ( a = 0.05 )，控制高考分数排名对录取概率的影响。假设高考分数排名每上升一个单位，录取概率增加 0.05。

参数 ( b = 0.1 )，控制学校专业招生人数对录取概率的影响。假设每增加一个招生人数单位，录取概率增加 0.1。

参数 ( c = -5 )，是截距项，用于调整整体的录取概率水平。但是我觉得这里是不是要分段做，因为不同排名段的人，这个a和b实在是差太多了吧

3.使用历史数据进行模型修改，采用最小化损失函数（如均方误差、交叉熵等）的方法来估计参数的值。

4.可以使用优化算法（如梯度下降法）来更新参数，使得模型的预测结果与实际情况尽可能一致。

模型拟合

定义损失函数：

在计算机上，你可以使用机器学习框架（如TensorFlow、PyTorch、Scikit-learn等）提供的损失函数，这些库通常包含了各种常用的损失函数，包括交叉熵损失函数。例如，在Python中，你可以使用以下代码定义交叉熵损失函数：

import tensorflow as tf

def cross\_entropy\_loss(y\_true, y\_pred):

return tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=y\_true, logits=y\_pred))

初始化参数 ( a, b, c )：

你可以使用随机初始化或者预定义的初始值来初始化参数 ( a, b, c )。在计算机中，你可以通过以下代码实现初始化：

import numpy as np

# 随机初始化参数

a = np.random.randn()

b = np.random.randn()

c = np.random.randn()

# 或者使用预定义的初始值

# a = 0.1

# b = 0.2

# c = 0.3

模型训练：

使用机器学习框架提供的优化算法来最小化损失函数。你可以选择梯度下降、Adam等优化算法，并在每次迭代中更新参数 ( a, b, c ) 的值。以下是使用TensorFlow进行梯度下降优化的示例代码：

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.01)

# 迭代训练

for epoch in range(num\_epochs):

with tf.GradientTape() as tape:

predictions = model(inputs) # 使用模型进行预测

loss = cross\_entropy\_loss(labels, predictions) # 计算损失函数

gradients = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(gradients, model.trainable\_variables))

# 输出当前损失函数值

print(f'Epoch {epoch+1}, Loss: {loss.numpy()}')

4.2 参数优化

停止条件：

在每次迭代中，检查损失函数的变化是否小于设定的阈值，如果是，则停止训练。以下是一个示例代码：

if abs(prev\_loss - loss) &lt; threshold:

break # 停止训练

prev\_loss = loss

调整学习率：

在使用梯度下降等优化算法时，你可以通过设置不同的学习率来优化模型的训练效果。以下是一个示例代码：

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.01) # 设置初始学习率为0.01

验证集监控：

你可以将一部分数据作为验证集，在每次迭代中使用验证集来监控模型在未见过的数据上的表现。以下是一个示例代码：

for epoch in range(num\_epochs):

# 训练模型

...

# 在验证集上计算损失函数

val\_predictions = model(val\_inputs)

val\_loss = cross\_entropy\_loss(val\_labels, val\_predictions)

# 输出验证集上的损失函数值

print(f'Epoch {epoch+1}, Validation Loss: {val\_loss.numpy()}')

* + 1. 选择模型

#todo:完善选择模型函数

其中：

a,b,c由拟合得出，为初始效用，由统计结果得出，由问卷偏好比例得出。

#todo:完善参数设定

涉及参数、

，为问卷调查统计偏好百分占比，，由主观偏差确定，初定范围为.

总方程满足：

=

* + 1. 模型验证

根据(3)所得的参数集，代入验证集数据进行验证，根据验证结果再次迭代(3)(4)步，直至预测准确率足够高

* + 1. 投入使用