**广 州 大 学**

**课 程 设 计 报 告**

**项目名称： 英语词汇量估算工具**

**学 院：计算机科学与网络工程学院**

**专业年级：软件工程2021级**

**成 员：32106300004 张景致 （组长）**

# 完成情况

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 身份 | 任务 | 任务占比 |
| 32106300004 | 张景致 | 组长 | 项目设计、实现、测试及报告 | 100% |

**已实现基本功能：**

1. 词汇库整理和词汇表生成
2. 简明词汇量估算算法
3. 无头式批处理界面
4. 桌面跨平台客户端
5. 自动对比测试模块

**额外实现功能（扩展功能）：**

1. 服务器-客户端模式
2. Web + WebAssembly客户端
3. Android HD客户端
4. 结果收集逻辑
5. 经验主义词汇量估计方法
6. 机器学习词汇量估计方法
7. 相似词汇资料库
8. 汉英选择题模式
9. 英汉选择题模式

**学生自评主要亮点：**

1. 基于不同算法思想的丰富多样的评测方式
2. 具备融合人工智能模型的算法技术
3. 多彩灵动-简明易懂的UI设计和实现
4. 具有可观难度的科学出题方法
5. 多种不同方向技术融合交汇的多元项目架构体系

**学生自评主要缺陷：**词库质量尚未达到业界较好水平

# 一、课程设计目的

本课程设计从解决相关应用领域实际工程问题出发进行综合实践，提高学生对软件工程发展的认识，强化学生软件能力训练，使得学生能够熟练使用专业知识和编程能力解决具体实际问题，完成软件项目的开发工作。学生通过团队合作应用软件知识开发一个具有实际工程应用背景的软件系统或解决实际工程应用背景的问题，进一步培养学生对软件工程中的需求分析等、系统分析、系统实现、软件测试的能力，掌握系统分析、系统设计、系统实现、系统维护等方法，加深学生对软件工程等相关课程的理解，增强对所选择的应用领域相关知识的认知，同时也进一步提高学生解决实际应用问题的工程综合运用能力。

# 二、课程设计内容

* 设计题目：按照软件工程思路设计 英语词汇量估算工具；

提交内容：

数据（如词汇表等）、算法思路、具体设计文档（报告）、代码等；

有实际创新加分

有扩展功能加分

分组责任参考（1-8人）：

总体设计；

算法设计（主要是词汇量测试算法和验证方法）；

前端选择和UI设计（web、桌面程序、app、小程序等都可以）。

简单数据库选择和设计；（不限定数据库）

演示测试：两种测试，一个GUI演示测试，一个是后台批处理测试。

报告撰写。 报告必须注明成员工作量分配（总和100%，如果5个人工作量一样的话就都是20%）

* 主要功能：

收集词汇列表等不同辅助数据，设计一至多种用户词汇量估算算法；

设计验证方法： 即 衡量你的算法， 估算出来的词汇量到底有多准确？

可与业内产品做比较比如：

http://testyourvocab.com/ （首选）

百词斩词汇测试

扇贝单次词汇测试

界面设计：可用web页面、桌面程序、app、小程序等。

后台批处理测试结果

可考虑 输入一个单词列表，直接算法后台计算结果

输入列表格式：词A， 认识； 词B，认识； 词C， 不认识；词D， 不认识；.... 。

输出结果：估算词汇量

界面实例测试结果

找不同学生，报告测试结果

主要数据：学号姓名（如有隐私考虑代号）、四级成绩、六级成绩、测试时间、测试结果。

* 扩展功能：

辅助数据和估算算法程序可以根据不同考虑放在服务器端或者客户端；

可用服务器端的数据、算法更新客户端相应的数据、算法；

发送学生测试实例结果到服务器端数据库

主要数据：学号姓名（如有隐私考虑代号）、四级成绩、六级成绩、测试时间、测试结果。

其他更复杂的改进，例如

同一服务不同客户端的交互（比如web和app端）

不同用户的交互（例如：单词对战）

**三、项目环境要求**

1、硬件需求:普通PC

2、软件需求:系统：Windows系统或linux系统（Ubuntu）

3、开发工具：不限

4、开发语言：不限

# 四、功能算法设计

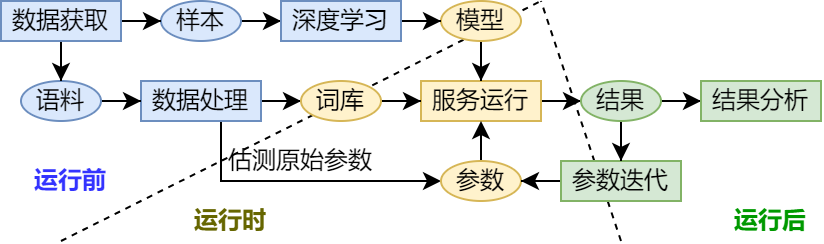
## 技术选型

因为这个项目将涉及服务器、客户端架构，需要图形化界面和数据统计、自动化网络测试等多样需求，判断需要将整个宏观项目拆分为多个不同规格、编程语言、项目管理平台和技术路线的项目，并在不同项目中采用相应方式进行沟通。综合各种需求、单人开发的局限性和当今的技术发展，选型如下：

|  |  |
| --- | --- |
| 需求或功能 | 技术 |
| 项目核心逻辑、数据处理 | Rust编程语言  Cargo项目管理  Tokio异步运行时  serde序列化标准 |
| 可靠的、正确的HTTP服务器 | Rust编程语言  Cargo项目管理  Rocket HTTP服务器框架  Tokio异步运行时 |
| 跨平台精致GUI和HTTP客户端 | Kotlin编程语言  Gradle项目管理系统  Ktor HTTP客户端  JetBrains Compose UI平台 |
| 数据分析、科学计算、深度学习技术 | Python编程语言  Anaconda包管理器  NumPy、SciPy数值计算库  matplotlib、seaborn绘图库  Jupyter Notebook平台  PyTorch深度学习框架 |

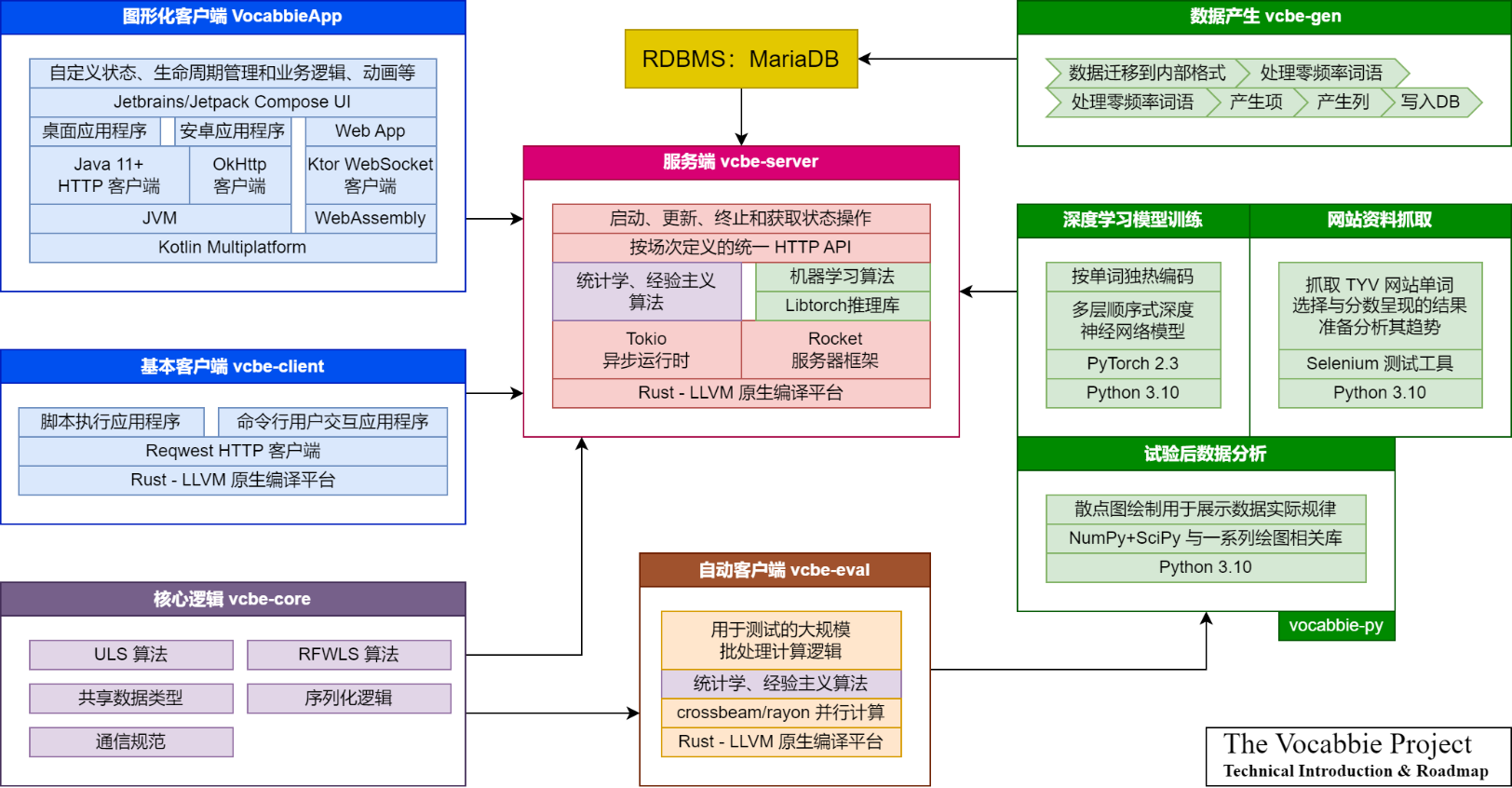
## 模块规划

依照数据发展变化的时间顺序来划分，该项目可以总结为包括运行前、运行时和运行后三个部分，故每一个模块也会和其相应的作用（属于的时间段）具有紧密关联，简单地可以用这一幅图概括这个项目的“生命周期”。



其中“数据获取”的任务囊括了人工和自动获取语料和运用自动化测试框架进行“爬虫”行为获取参考网站（test-your-vocabulary）的样本的两个不同操作，语料产生出词库，样本用于深度神经网络训练产生推理模型。值得注意的是，虽然“运行时”部分只有“服务运行”一个操作或任务，但却包括了这个项目的大部分业务逻辑，因为所有的服务端和客户端都包含在其中。再者，通过服务运行可以得到结果（词汇量预测），经由各种分析手段处理可以得出相应结论，进而对服务运行使用的参数进行优化，而优化后的服务运行得出的结果又是不同的，这里形成了一个参数迭代的循环。

## 模块设计

结合前面的技术选型工作，可以对模块规划进行一个拓展，形成一个具体可行的设计方案。方案根据每个模块的不同，覆盖了项目的方方面面，也会形成模块之间的各种数据流动和信息交换，概括起来如下图。

根据主要用途和面向的用户这个详细设计可以被分为几个部分：蓝色的面向最终用户的客户端部分；红色的在服务器上运行的服务端部分；绿色的数据处理和统计计算部分；棕色的测试部分和紫色的核心逻辑部分。

## 模块：数据产生

作为一个词汇量测试软件，一个完整的词库是整个项目的基础。基于模块设计中对测试方式的看法，得出该项目中的词库其中每一项除了ID和单词本身，需要几种不同的信息：

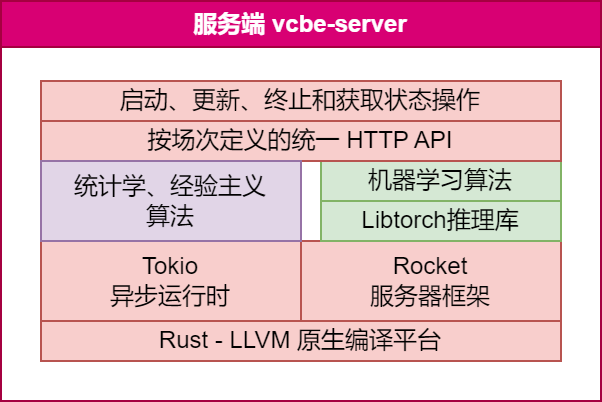
* 词频 可以是词语在该词汇表对应语料库中的每一个语料单元的出现次数，也可以是出现概率，只要可以代表该词语的“流行性”足矣。
* 词义 该词语的所有（常用的）不同意义的集合。
* 近似词 同该词语计算按从莱文斯坦距离[[1]](#footnote-1)小到大的顺序排列的前n个近邻词语的集合。
* 包含词 包含该条目对应的词语的词语的集合。
* 被包含词 同上，但是关系相反。

其中词频的作用是显而易见的，这将被作为估计算法其中“难度”指标的替代属性使用；而余下的几个属性将作为选择题的出题材料使用。

本项目的词汇表基于从“网易有道词典”网站抓取的公开数据来形成词汇表为了产生这些数据，包含过多、杂乱资料的原始词汇表需要被处理才能够使用。同时，一些较为繁重的计算任务也在这一步可以预先处理，达成一种提前缓存某些算法结果的目的，减轻服务端处理包袱。经过长时间的摸索实验，得出以下数据预处理和算法结果缓存的步骤：

* 从抓取的原格式（API回复JSON格式）转移到serde兼容内部格式，并使用MessagePack格式重新序列化，提高后续处理性能
* 过滤数据无意义或不合法（频率为0）的条目
* 根据词频的log10值（数量级）定义每一个词语的级别，每0.5范围定义一个区间
* 对每一个条目（单词）：
  + 计算与其他所有单词的莱文斯坦距离，并得出最低的n个
  + 计算与其他所有单词的包含和被包含关系
* 收集当前版本项目用到的数据
* 将条目写入MariaDB数据库

## 模块：服务端



服务器会同时处理所有客户端的请求，运用各种算法和外部库承担计算工作。因此，采用Rust、Tokio和Rocket以最大化可以实现的安全性和性能，避免后期实现其他模块的时候频繁调试HTTP API。

为了同时为多个不同客户端服务，服务器将每一次测试量化为一个带有ID的会话（Session）并在全局会话池中储存，每一个客户端理论上可以不受冲突地访问服务器提供的HTTP API。服务器具有简单地几个URL终点：

* /index 展示帮助引导网页
* /start 启动会话
* /state 查询会话状态
* /submit 提交操作

其中start操作通过kind参数指定会话类型，state和submit操作指定id确认目的会话。只有start和submit操作具有副作用。机器学习模型在需要推理的时候按需加载，在此之后留存在内存等待继续使用。

为了防止过多未终止的会话导致服务器产生问题，每一个50分钟内没有任何端点访问的会话将被清理。

## 模块：图形客户端

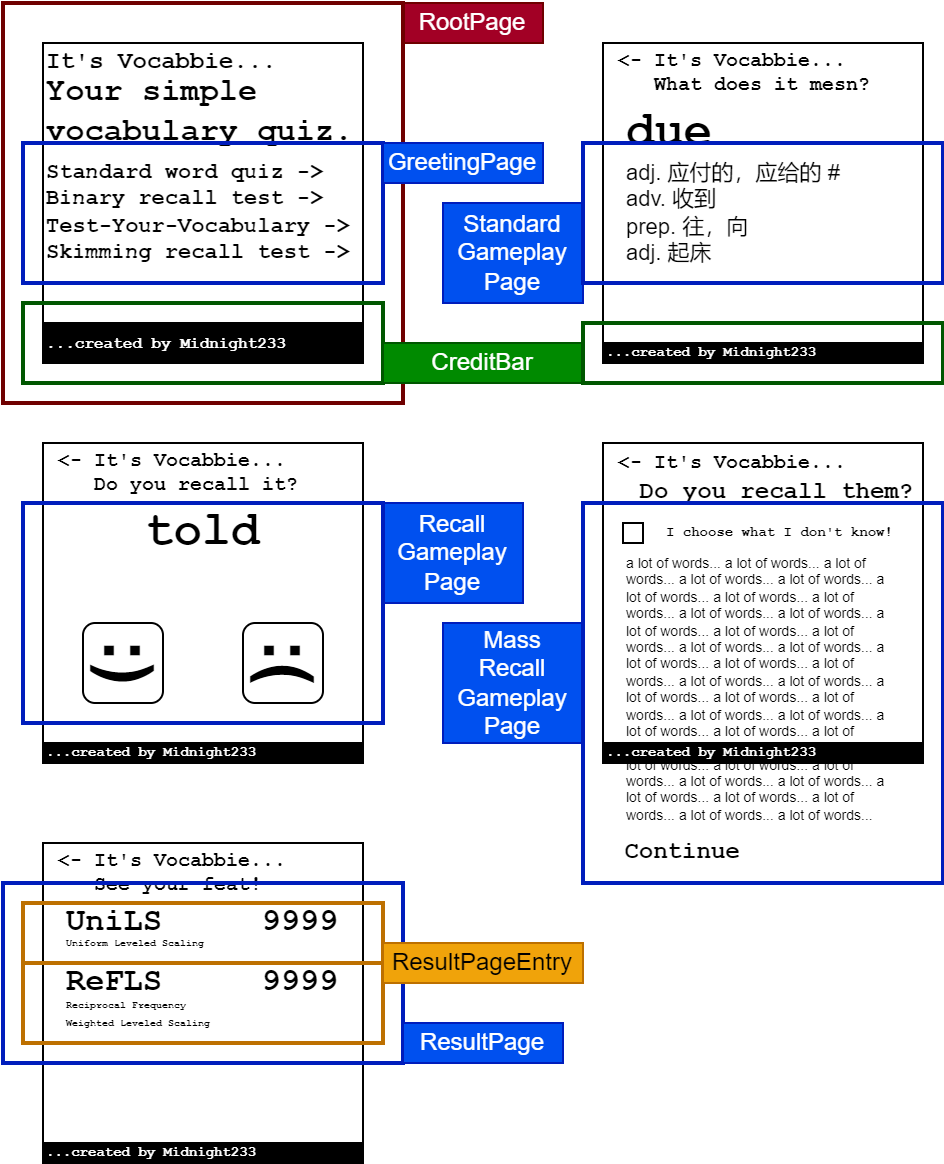


我将这个项目的客户端定位为“实用性和娱乐性兼备的非专业软件”，故希望为它实现较为生动活泼、美观大方的用户界面，这样不可避免地需要进行艺术上的和人体工学上的一些设计。我从PANTONE色卡的设计上获得些许灵感：

 🡨 **PANTONE色卡：实用性为主，但也有设计**

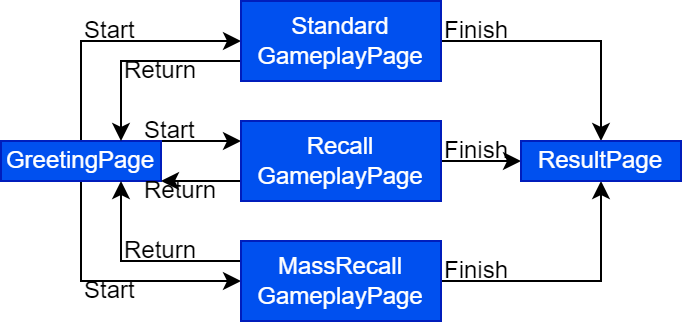
在兼顾设计感（还有品牌宣传）的同时，色卡让人可以集中注意力到其中的内容。因此我利用Compose跨平台UI框架设计了“卡片式”的应用UI。在尽可能实现画面元素的重用的基础上，实现了每一个不同UI之间的顺滑过渡效果。好的动画效果可以使得应用程序的气氛更加柔和，更加具备人文气息。

UI的整体设计可以从这幅图看出：（见下页）



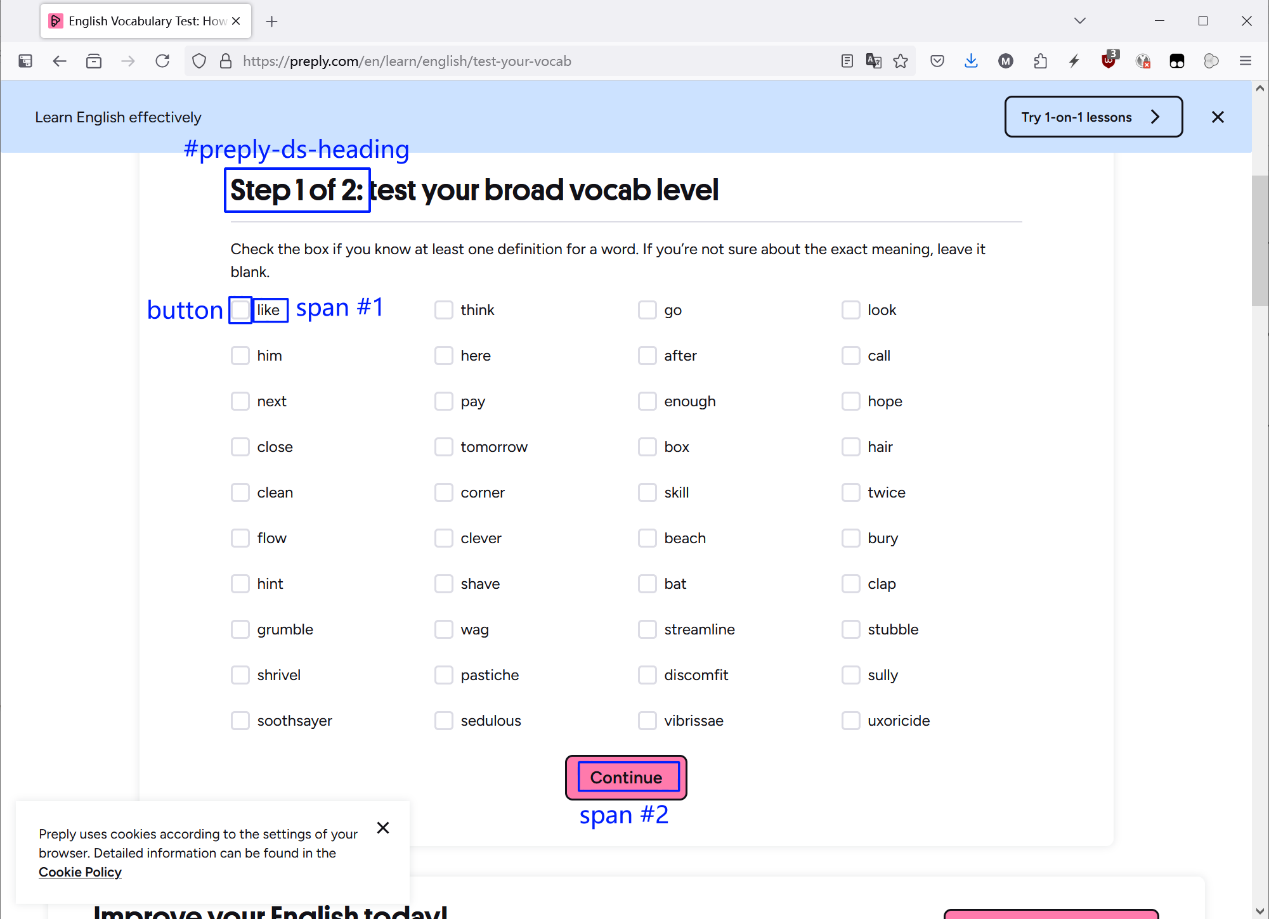
UI的最外围是用于实现“卡片”效果和包含其中（跨页）固定元素的容器RootPage（红）。值得注意的是，标题和子标题字体大小都可以随页面变化而动态变化。页面的主题部分包含各种Page（蓝）吗，其中包括了该页面实际上（除去装饰性和固定元素）实际上所需要展示的内容。其中MassRecallGameplayPage页面长度大于“卡片”，故引入滚动机制。ResultPage页面由ResultPageEntry元素组成。

为了最大程度降低状态管理的复杂性，画面的转移经过设计之后可以呈现出清晰的规律，不会存在“交通枢纽”（页面交织）的情况，便于拓展和维护：

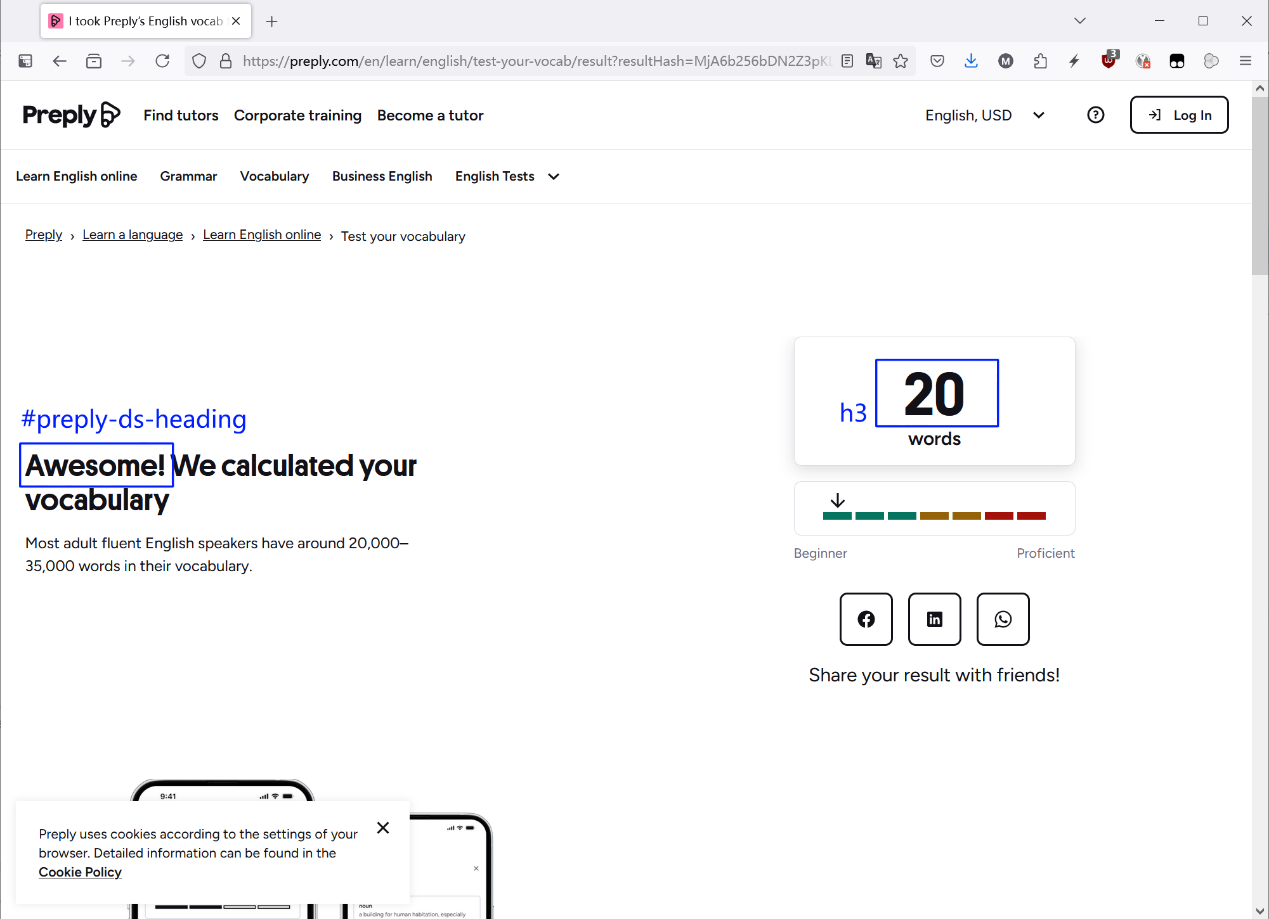


## 模块：网站抓取

使用Python和Selenium自动测试框架编写脚本。

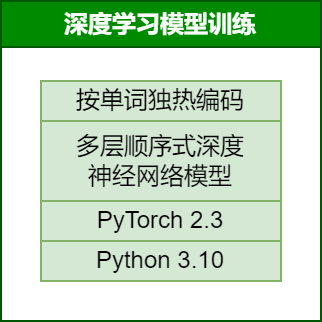


找到带有“preply-ds-heading”类的元素可以确认当前处于哪一页，寻找ID中包含“word\_”的按钮，并根据预先决定的正确率进行点击操作，并查找包含“Continue”字样的span元素点击进入下一页。

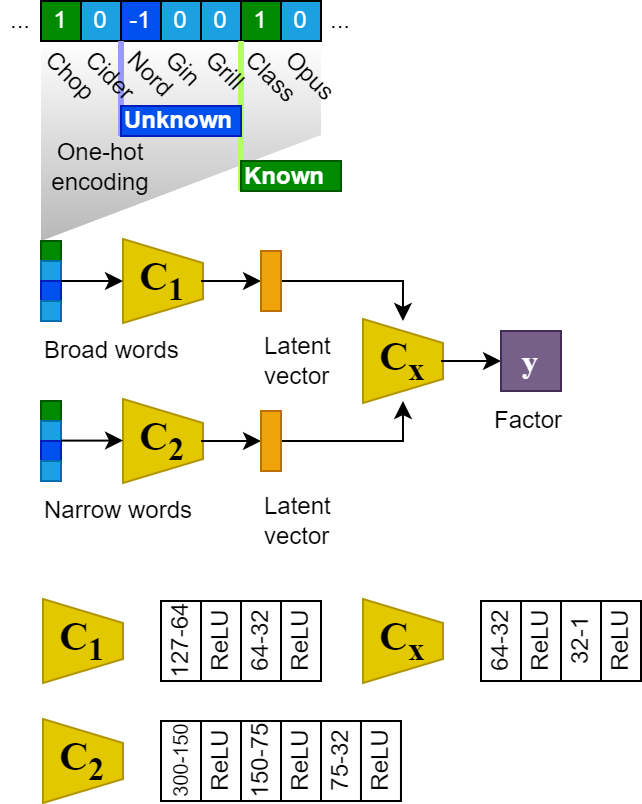


在结果页寻找h3标签确认成绩。

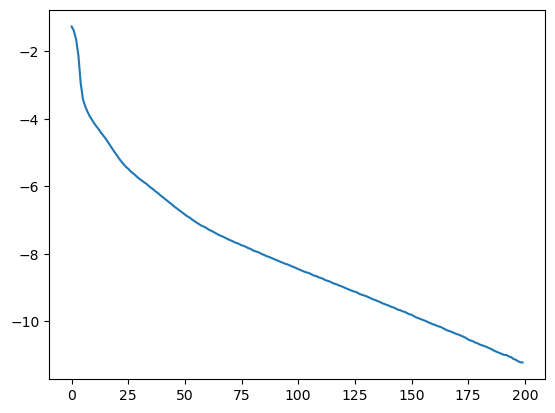
## 模块：机器学习



根据来自上一个模块的分析结果可以得知Preply网站考察的单词范围是较为有限的。以此猜测单词出现与否、选择与否与最后展示的分数具有决定性的关联，故分为broad和narrow单词分布进行独热编码，认识的单词为1，不认识的为-1，未遇到的为0，构建复合式深度（非卷积，因为推测近邻的元素没有必然联系）神经网络并进行训练。模型架构如图所示：



使用学习率为1e-5的Adam优化器进行200轮训练，效果较为显著，并且具有继续训练的一段空间。为防止过拟合现象，故不继续训练。

**🡨 LOSS (LOG) 图像**

## 估计算法



首先明确一下我们的目标：估算词汇量——也就是估计“测量词汇量”的结果。实际上，测量词汇量是不可能做到的，因为英语词汇理论上是无限多的，我们只能测量某个词汇表的词汇，这样就引出了我们的首要两个假设：

* 假设A 估计的“词汇量”就是我们的目标
* 假设B 测量词汇量 = 测量相对词汇表的词汇量

较为遗憾的是，“词汇量”这个概念本身就不甚清晰。如何才算一个词汇在某人的词汇量中呢？我们在这个项目直接询问“知道与否”或“是否答对”。

* 假设C 主动回答“知道”或者答对选择题的单词算入词汇量

在数据处理的模块中可以知道该项目的词汇表由多于60000个不同的词汇。在情理上和现实情况上来说，这使得“测量”操作无法完成，这就是“估计”词汇量的意义所在，也就是说要通过测量少于60000个词汇来明晰对这60000个词语的词汇量，其中每一个单词需要反应某一些单词的掌握情况。我们一般认为难度相近的词汇可以相互代表。

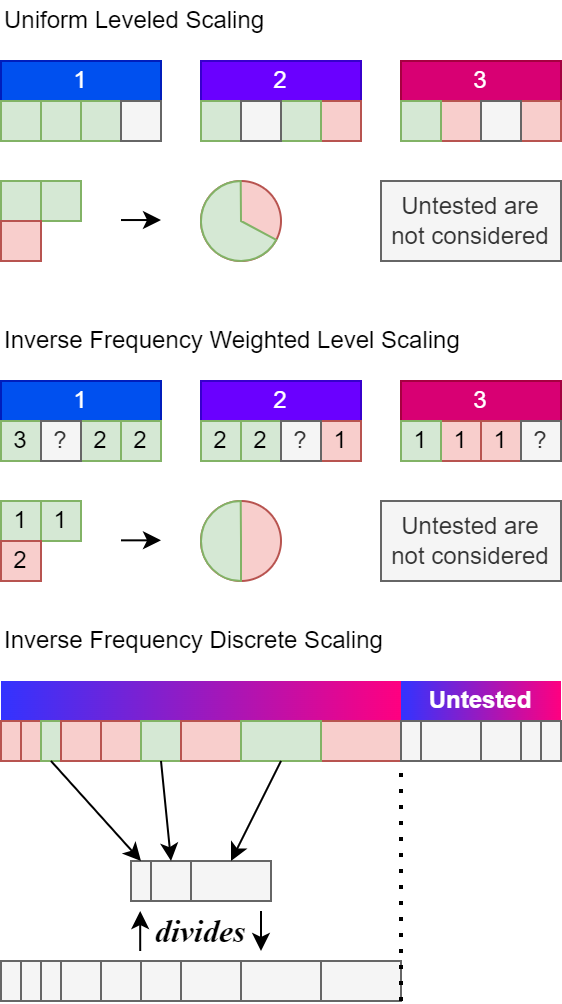
* 假设D 一个单词可以代表多个难度相近的单词

在这个项目中，我将“难度”和“稀有程度”等同起来，作为同难度词语的衡量标准，用于构建估计算法。

* 假设E 频率的倒数算作单词的“稀有程度”
* 假设F “稀有程度”=“难度”

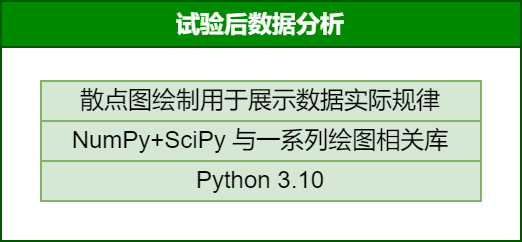
***如此繁杂的假设将会使得估计结果难以准确，但实际上不存在准确的估计结果，所以我认为估计结果可以反映个体差别就可以算作有效。***

在之前的模块中，根据词频的log10（数量级）数值进行分级，这可以引出项目中主要使用的基于统计学的三种不同的算法：

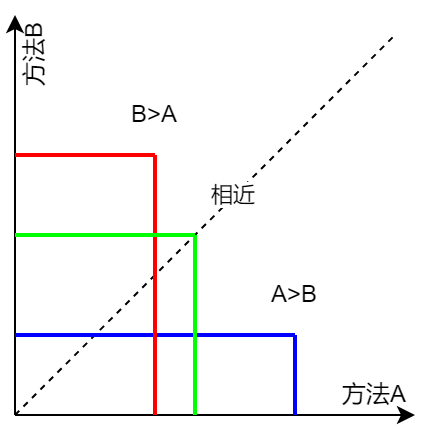


其中ULS（1）算法不考虑同一级别内不同单词的词频，通过每一个级别内的正确率来估计该级别词语的掌握率；RFWLS（2）算法在ULS（1）的基础上使用词频的倒数作为权重使得出现频率较低（较稀有🡪较困难🡪较有代表性）的词语具备更高的地位；IFDS（3）算法忽略级别的影响，通过计算答对的情况的权重之和和所有题目的权重之和的比值来估算。

## 评价算法



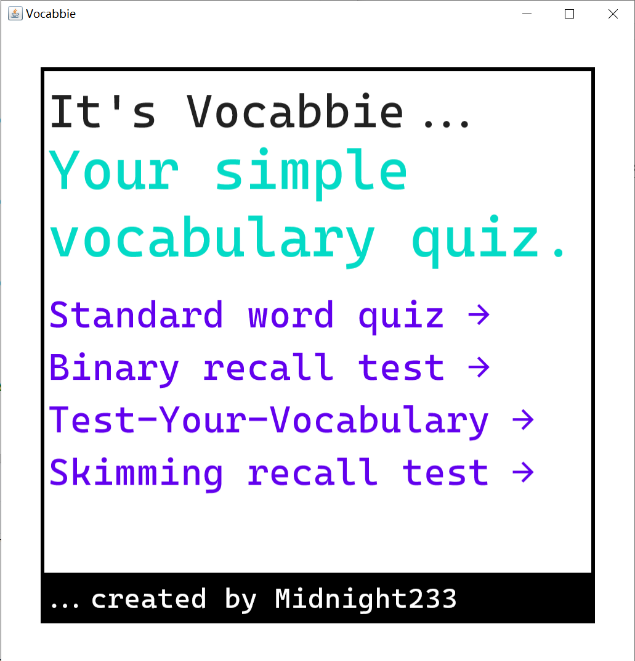
由于客观上参考网站Test-Your-Vocabulary（以下称TYV）和该项目的给分范围存在比较大的差异，故基于加减的统计学手段是较为难以反映实际情况的，据此我设计使用二者之比和二者成一点的值来设计对比评价，使用散点图可以直观地看到二者之间的关系。

 🡨 示例散点图

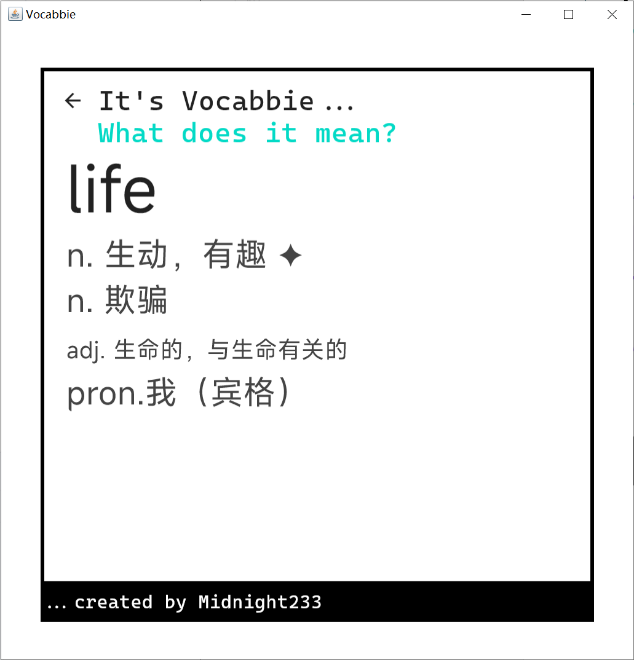
直观上来说，成绩与TYV愈接近愈能证明该项目算法的可用性，但我仍然希望强调TYV也不可能是一个“准确”的标准，只是我们为了验证这个简单的项目而定下来的验证手法；除了散点图，还会有一些其他的分析手法出现。

# 五、模块展示说明

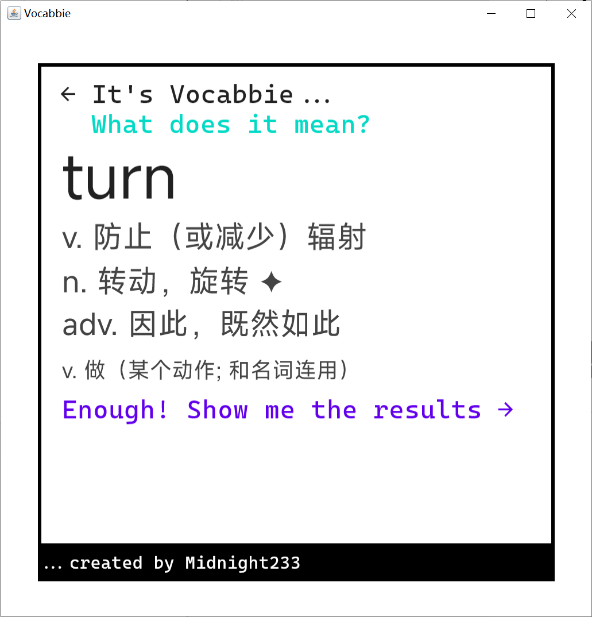
## 展示：图形客户端

 🡨 开始页面

在这个界面上，用户可以选择想要使用的模式。

 🡨 选择题页面

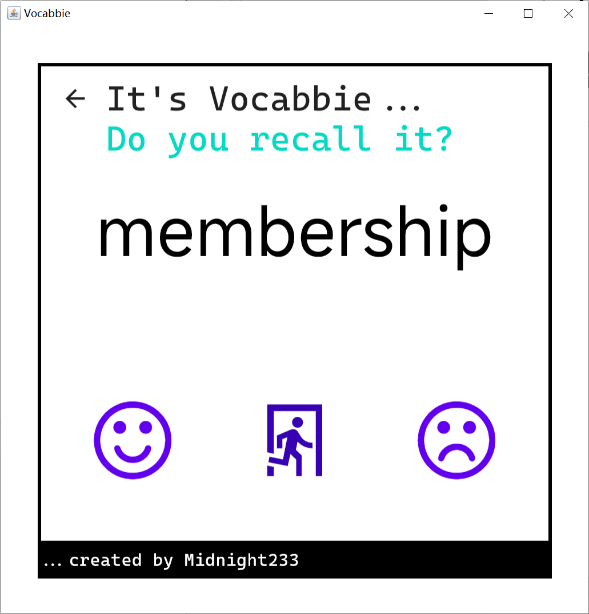
这个模式中，用户需要选择正确的选项来证明自己知道这个词语。题目和选项均会根据实际情况调整大小避免出现显示溢出的情况。

 🡨 选择题页面，包括结束按钮

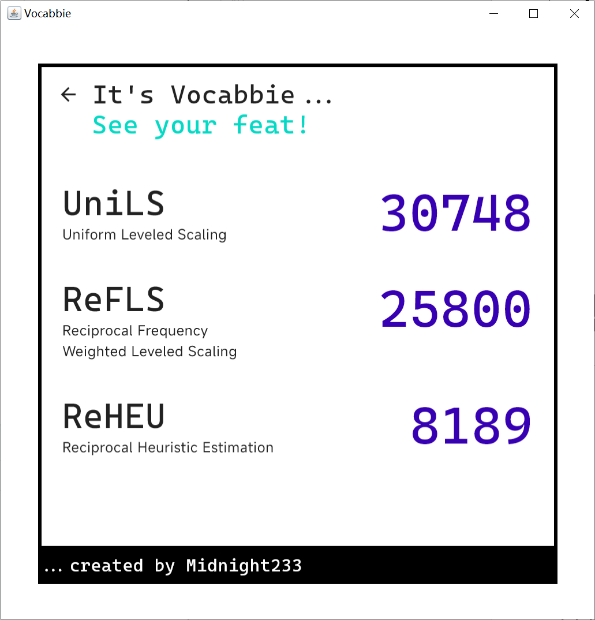
最少完成24个题目在之后会展示这个选项。

 🡨 回想测试页面

通过表情图标按钮选择知道当前单词与否。

 🡨 回想测试页面，包括退出按钮

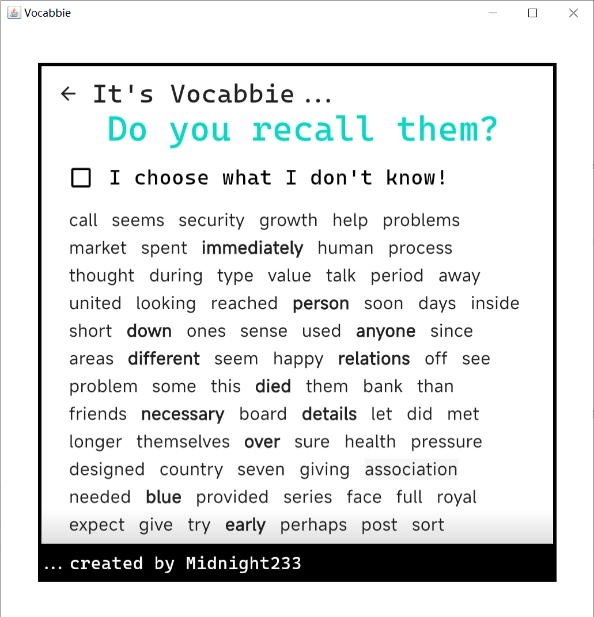
选择“紧急出口”图标即可结束测试。

 🡨 结果页面

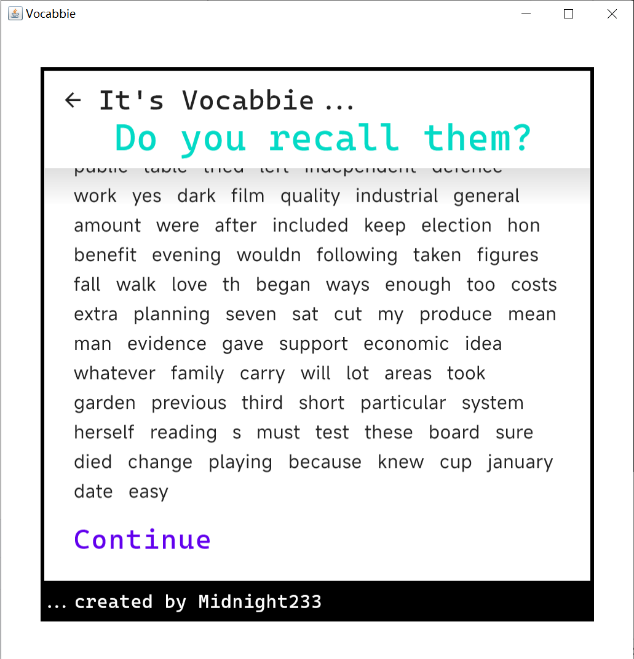
可以查看使用不同方法计算的词汇量，获得不同的参考值。

 🡨 结果页面，使用机器学习模型

使用深度学习算法的模式是独立的。

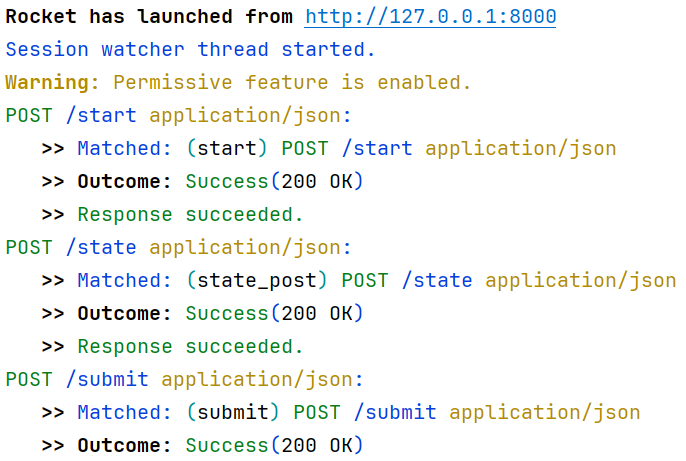
 🡨 批量回想页面

可以根据上面的复选框选择要在下面选择知道的还是不知道的词语。

 🡨 批量回想页面，底部

选择完成（滚动到底部）之后可以选择继续，退出按钮会在8批后出现。

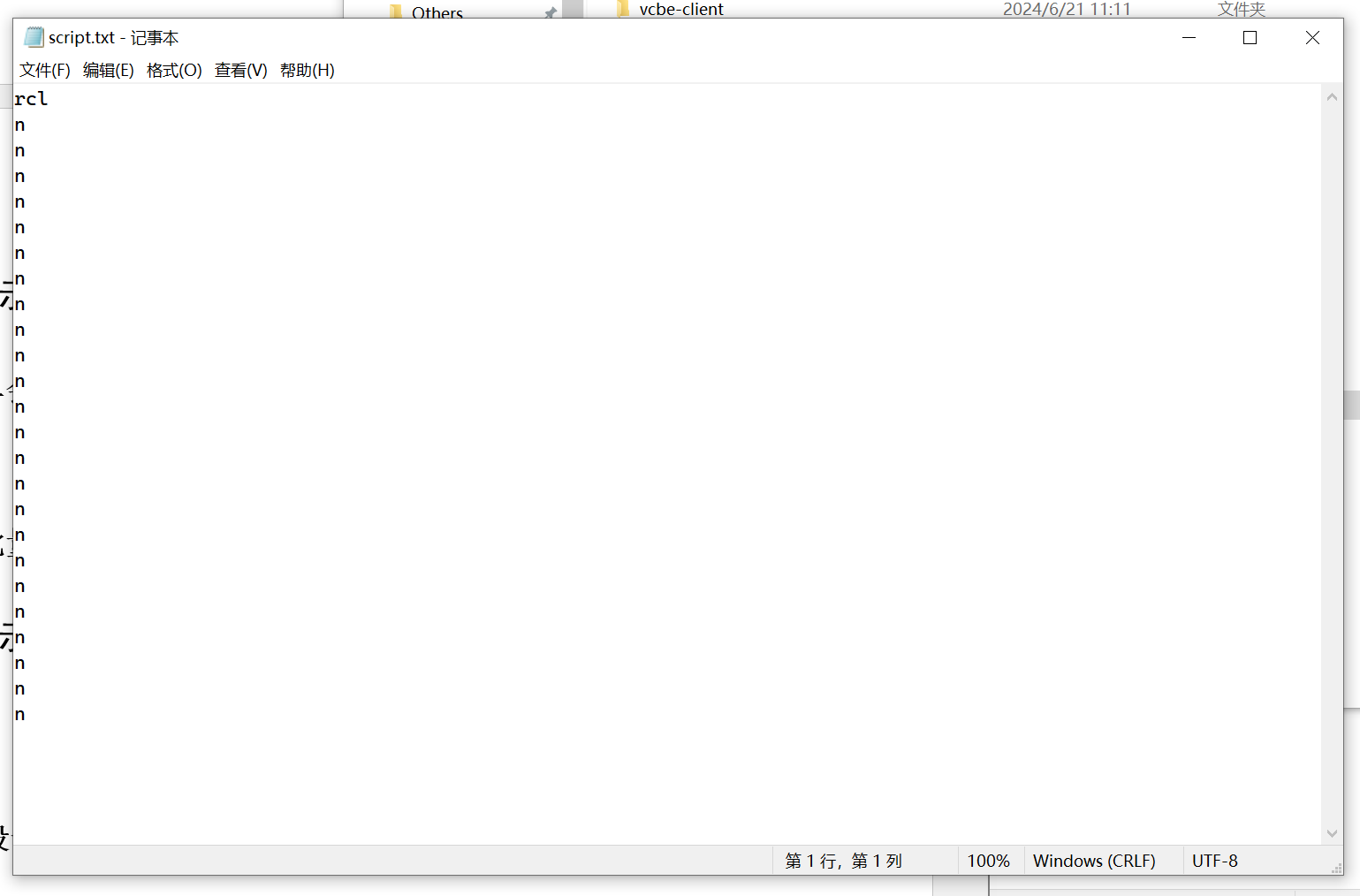
## 展示：服务端



（调试模式的）服务端，其中日志的第二和第三行是会话清理线程启动的提示和服务器被按照“松懈模式”（透露选择题答案）模式编译。

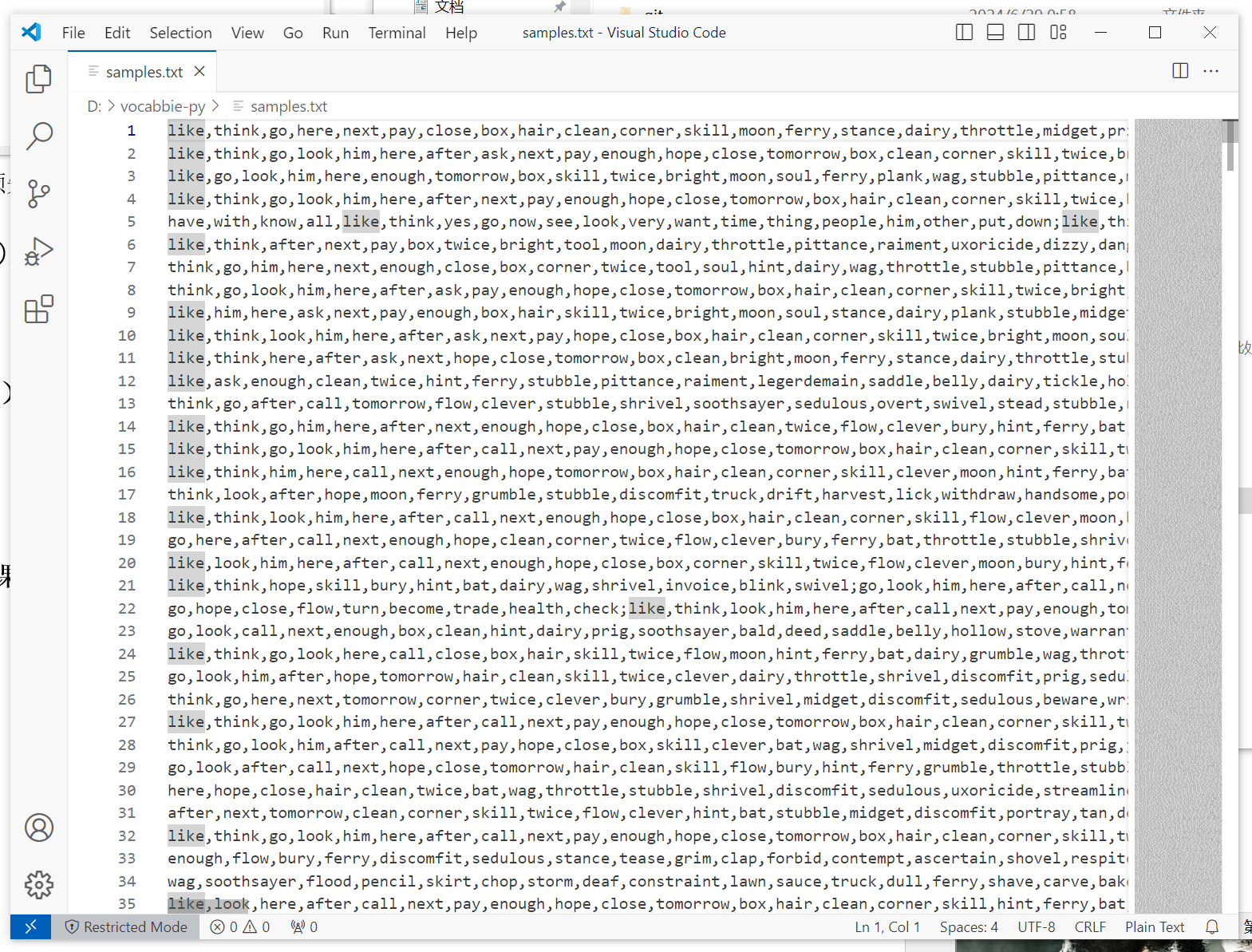
## 展示：批处理

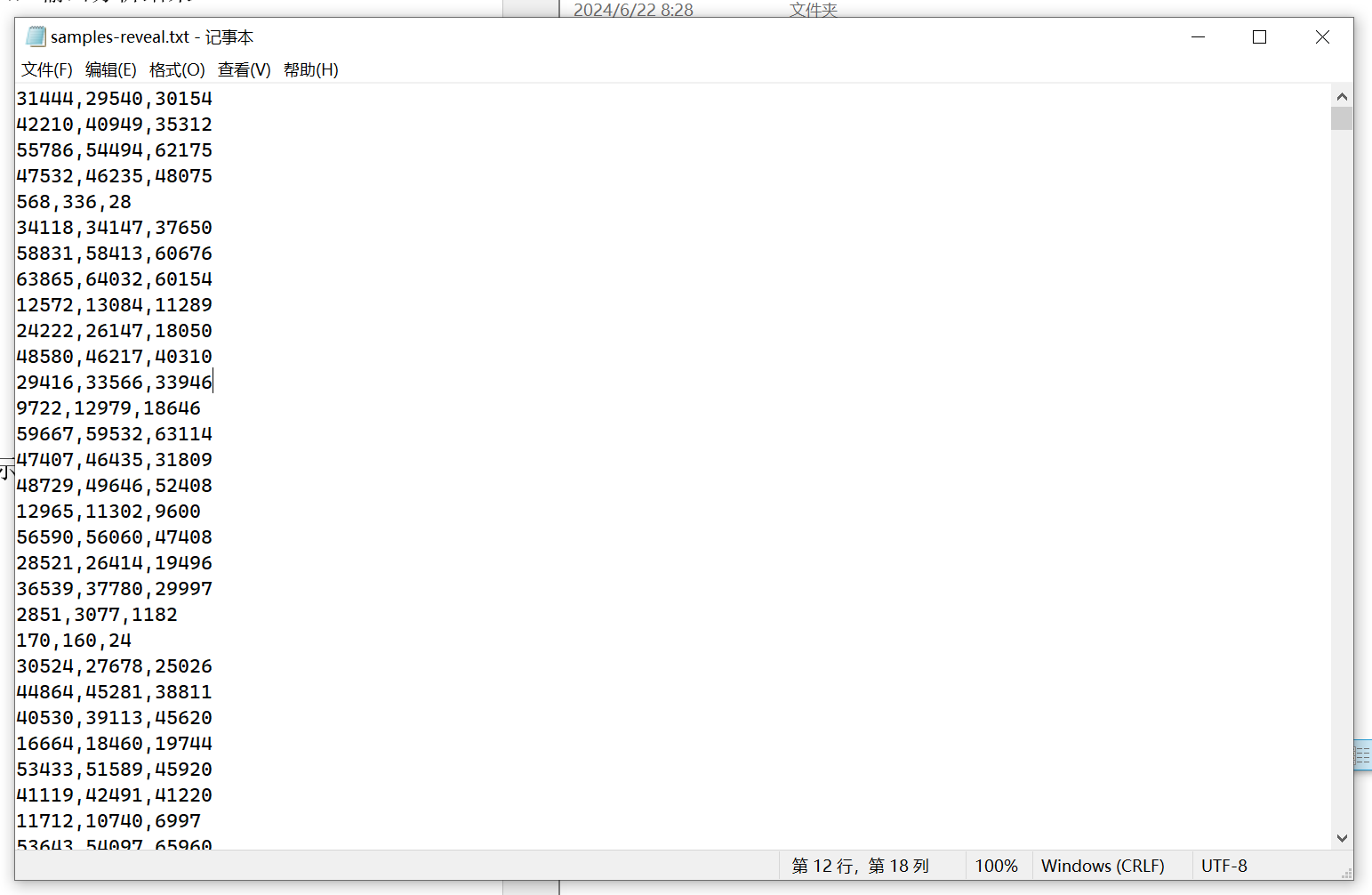
### 单会话脚本

 🡨 脚本展示

根据预先决定好的“正确与否”设计来替代人工输入，输出分析结果。

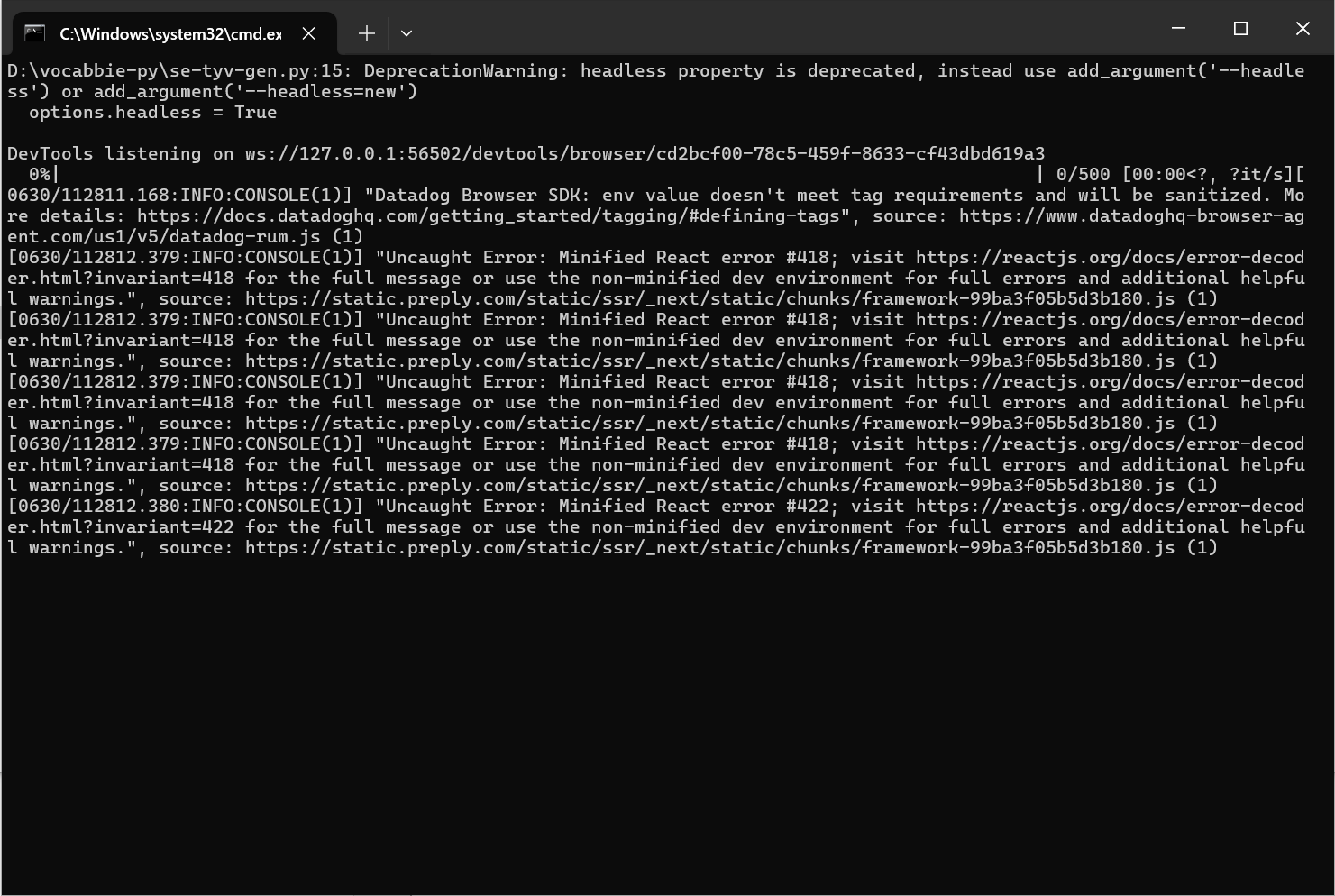
### 批量分析

 🡨 脚本展示

 🡨 输出展示

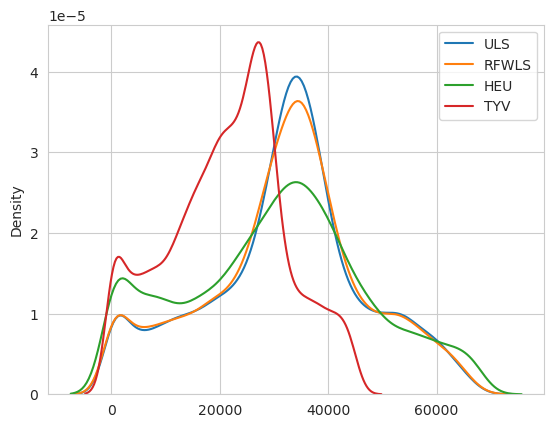
给出知道、不知道的词语批量通过多种算法分析，给出分析结果。

## 展示：网站收集



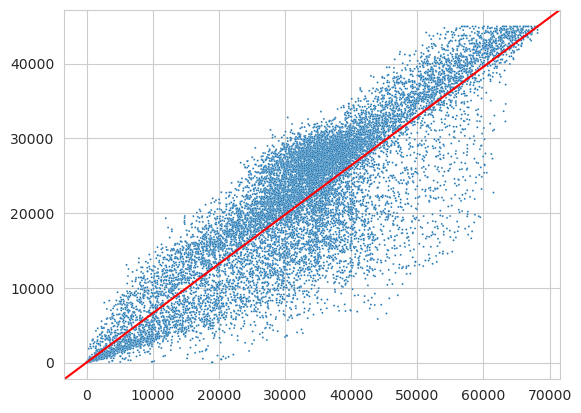
使用预先准备的脚本，可以并行多次爬取网站选择和结果，自动决定选词内容和正确率，记录必要数据并序列化存储。

## 结果分析



上图是不同算法（使用同一批样本）导出的分数的概率分布拟合曲线。可以看到我的三个算法（ULS、RFWLS、HEU/IFDS）都呈现出和参考算法（TYV）相似的规律：高低双峰和类似正态分布的整体构成。可以发现TYV的数值范围最大到42000左右，而我的所有算法范围最大到64000左右。

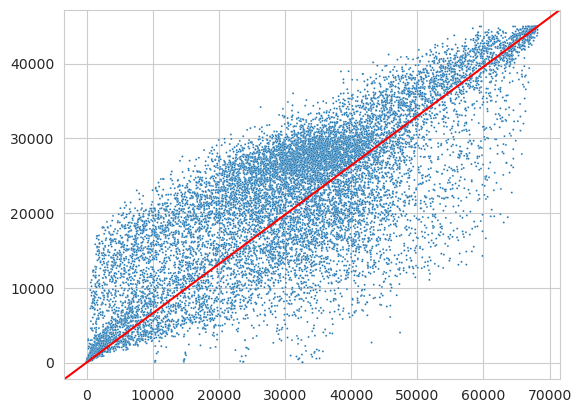
就我的算法来看，ULS和RFWLS的区别并不是很剧烈，不同点主要集中在高峰处。而HEU/IFDS算法的成绩分布更加均匀，低分数的密度尤其更高，和TYV的预测比较相似。同时，HEU/IFDS算法更容易打出比较高的分数。



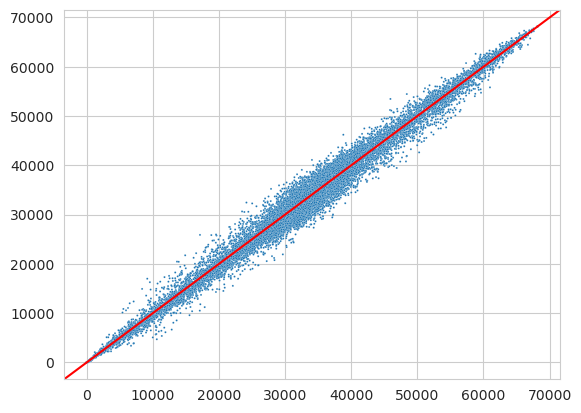
上图是RFWLS（x）和TYV（y）的对比，红色的线代表二者在相对给分范围的比例上相同的点集合。可以看到整体上二者是具有强烈联系的，尤其是在分数趋于中等水平（也就是密度最高的区域）上的时候。

RFWLS打分悲观于TYV的情况时有发生，但总体来说程度不是很大而且十分稳定。而打分乐观于TYV的情况虽然频率上更低，但是程度范围似乎显著广于前者，说明对于某些频率较低的词语来说两个算法的看法有所不同。

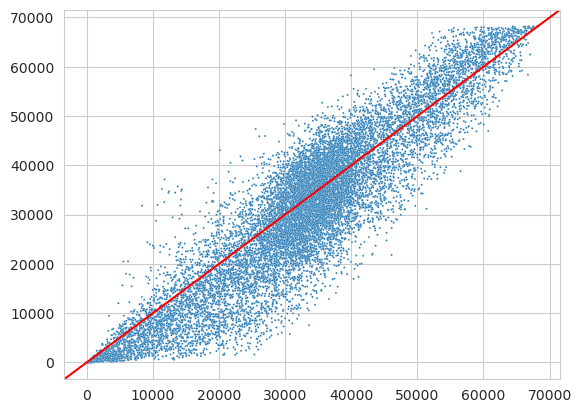
值得注意的是，RFWLS作为计算手法上更加精确的ULS版本，其性能实际上没有本质的提升，故不再展示其统计图表。



上图是HEU/IFDS（x）和TYV（y）的对比，红色的线代表二者在相对给分范围的比例上相同的点集合。可以看到整体上二者是具有强烈联系的，但是于RFWLS不同的是，IFDS在低分段常常比TYV更加悲观，而在中等分段更加乐观的概率又上涨了。这里说明两个算法对每个单词权重的计算处理或者分配策略可能存在较为显著的区别。



上图是ULS（x）和其更加精确的版本RFWLS（y）的对比，可以看到二者差别主要集中于中等分数段，与第一幅图显示的情况相符。



上图是RFWLS（x）和IFDS（y）的对比，可见有无分级主要对中间分数段的影响较大，而在低分段RFWLS相对IFDS偏向乐观，高分段偏向悲观，而IFDS相对来说要更加平稳，对不同的用户更有可能获得较为相似的分数。

## 讨论与结论

# 六、课程设计体会（个人体会）

1. 又称Levenshtein距离，是编辑距离的一种。指两个字串之间，由一个转成另一个所需的最少编辑操作次数。 [↑](#footnote-ref-1)