面向真实科学问题的Java课程设计

专业：计算机科学与技术
姓名：李春良
学号：1120193226

1. 项目简介

简单地说，本项目要实现的最终目标是，判断某蛋白质序列是否是DNA结合蛋白。报告给出的具体实现已经实现的结果是，将给定的蛋白质序列FASTA格式（序列名/注释+单个字母表示的核酸或氨基酸）输入，转换成一串便于分类器理解的向量。向量间数学特征的差异，可以反映不同序列输入的文字特征和语义特征，有助于区分不同的蛋白质序列。

项目初步实现的功能包括，运用Kmer分词方法创建词典，给出了包括One Hot Encoding、Bag Of Words、TF-IDF在内的三种特征提取方法。用户可以通过有限的图形交互界面来实现基于文件输入的交互，并获得向量序列的文件输出。

项目简单实现了数据与表现的分离，提供了一定的可扩展性，既可以通过调用项目实现的有限内容来获得向量序列的文件输出，也可以通过调用项目包含的底层数据逻辑来构造进一步的机器学习模型。

2. 实现方法与细节

注：benchmark_set 样本集需要用户拆分成两个文件，即 DNA 结合蛋白集与非 DNA 结合蛋白集，前者本文称之为正样本(集)，相应后者就是负样本(集)，independent_set 则称之为独立样本(集)。

I. 输入模块（样本模块）

输入类型	含 FASTA 编码蛋白质序列的简单文本文件(*.txt)
输出类型	管理了蛋白质序列的序列库 SeqLib 对象

包含的package：com.jbji.sample, com.jbji.sample.io com.jbji.sample.exception

这个模块定义了样本对象的结构和样本对象管理器，以及管理管理器及其附加信息的样本库，同时提供了输入输出方法，并定义了一些异常类。

Class UnitSeq

java.lang.Object
com.jbji.sample.UnitSeq

描述一个蛋白质序列最基本的方法是一个 FASTA 单元，同样，设计 UnitSeq 类的想法，就是通过组合 description 和 sequence 两个 String 对象，来实现对基本 FASTA 单元的存储。

Method Summary

All Methods

Modifier and Type	Method	Description
boolean	contains(java.lang.CharSequence s)	检查序列是否包含某个字串
java.lang.String	getDesc()	获得描述
java.lang.String	getSeq()	获得序列
int	hashCode()	
void	setData (java.lang.String _description, java.lang.String _sequence)	设定当前序列单元的描述和序列
java.lang.String	toString()	

Class Unit

java.lang.Object
com.jbji.sample.Unit

Unit 单元类的设计想法是，有了 UnitSeq 类，又一个很自然的想法就是把这些原始序列和它们的编码结果存放在一起，由于考虑到编码结果可能不止一种类型，因此利用 HashMap 容器来存储编码器类型(EncoderType 类)对应的编码向量(CodeList 类)。我们稍后会介绍 EncoderType 类与 CodeList 类的设计。

Method Summary

All Methods

Modifier and Type	Method	Description
void	addEncode(EncoderType e, CodeList u)	添加编码
java.lang.String	getDesc()	获取描述
CodeList	getEncode(EncoderType e)	获得编码
java.lang.String	getSeq()	获取序列
void	resetEncode()	重置编码
int	hashCode()	

Class CodeList<T extends java.lang.Number>

java.lang.Object
com.jbji.sample.CodeList<T>

All Implemented Interfaces:

java.lang.Iterable<T>

CodeList 类的设计是为了存储单个单元的单次编码结果，或者叫编码向量，因此它在设计上是 Unit 的一个可扩展部分，为了提供可能存在的整型或浮点型编码向量结果的保存，类使用了泛型模板。

本质上本类是包装了一个 ArrayList 对象，通过提供迭代器的方式允许实现对编码结果内每一个单元的单独访问。

Method Summary

All Methods

Modifier and Type	Method	Description
int	getFreqSum()	获得序列中所有元素的加和
int	hashCode()	
java.util.Iterator<T>	iterator()	迭代器
int	size()	List 的大小，即包含元素个数
java.lang.String	toString()	快速获得编码字串

我们还没有介绍 EncoderType 类，读者现在只需要知道，这个枚举类定义了一系列编码器类型枚举量，便于编码方式理解和表示。

至此，通过 Unit 类组合一个 UnitSeq 对象和一个 CodeList 哈希容器，我们已经实现了对单个 FASTA 序列单元的存储与表示，以及对其编码结果的存储与表示。

至此，我们已经可以设计输入的获取方法了。

Class SeqGetter

java.lang.Object
com.jbji.sample.io.SeqGetter

SeqGetter 被设计用来从一个文件 File 或者一个 fasta_str 序列字符串直接获取数据。

在调用构造函数时，就会利用正则表达式去匹配输入中的所有符合标准格式的 FASTA 字串，并将其保存到内部的 Matcher 中。

用户在获得匹配结果时，可以不断调用 nextProtein(UnitSeq s)方法，该方法会将完整匹配结果作为返回值返回，同时将描述和序列传给传进来的参数 UnitSeq s，这样通过一次次调用 nextProtein 方法，直到返回结果是 null 代表所有的匹配结果都已给出完毕。

实际使用该类时，尽管用户也可以调用该类中的具体方法，但更推荐将该类作为后续序列管理器构造的一个参数来使用，也就是让另一个类来管理具体输入并直接将输入转换成序列管理器中的有限个序列单元 Unit 对象。

Constructor Summary

Constructors	
Constructor	**Description**
`SeqGetter(java.io.File file)`	
`SeqGetter(java.lang.String fasta_str)`	

Method Summary

All Methods

Modifier and Type	Method	Description
`java.lang.String`	`nextProtein(UnitSeq s)`	

Class FileSaver

java.lang.Object
com.jbji.sample.io.FileSaver

顺便介绍保存文件的类，目前该类只实现了一个简单方法，就是将字符串 String 对象中的文本内容保存到指定文件，具体调用依赖于用户的设计。具体来说，就是用户通过调用序列管理器的获取各类信息字符串接口来获得要保存的信息，再指定保存路径去保存文件。

Method Summary

All Methods

Modifier and Type	Method	Description
static void	save(java.lang.String to_save, java.io.File file_name)	保存函数，将字符串保存到文件

Class SeqMan

java.lang.Object
com.jbji.sample.SeqMan

All Implemented Interfaces:

java.lang.Iterable<Unit>

前文说到，我们有了输入类 SeqGetter，但并不希望用户直接使用这个类。此外，对于管理输入结果的类，我们还希望它可以直接将输入转换成序列管理器中的有限个序列单元 Unit 对象，这就是 SeqMan 类。

调用 SeqMan 类，你需要给定一个 SeqGetter 输入序列，SeqMan 会将其转换成 Set 容器中的 Unit 对象。SeqMan 是允许被迭代访问的，这就意味着 SeqMan 本质上就是打包了 Unit 对象，这些 Unit 对象是可以直接迭代访问的。

因此，对于每一个给定的序列输入文件，实际上都直接对应着一个 SeqMan 对象，要管理多个文件，只需要将这些 SeqMan 对象组合，这就是接下来要引出的 SeqLib 类。

Constructor Summary

Constructors	
Constructor	**Description**
`SeqMan(SeqGetter getter)`	利用输入来建立一个序列管理器

Method Summary

All Methods

Modifier and Type	Method	Description
`java.util.Iterator<Unit>`	`iterator()`	提供管理器内单元访问迭代器
`int`	`size()`	管理器内单元的数量
`java.lang.String`	`toString()`	

Class SeqLib

java.lang.Object
com.jbji.sample.SeqLib

SeqLib 是整个输入模块的核心之一，也是连接向量化模块的桥梁。

序列库 SeqLib 最开始的想法是管理多个 SeqMan 对象，即管理多个序列输入文件的内容。因此，构造函数当然应该包含两种可能性，已经建立好的 SeqMan 对象，或者干脆将 SeqGetter 给序列库，让序列库自己来建立序列管理器，这里选择了后一种方案，构造函数就是以 SeqGetter 作为属性的。

管理 SeqMan 本身，主要是提供获得序列管理器 SeqMan 的方法，这些信息是编码器和词典在具体调用时必须要获得的信息。

除此之外，SeqLib 管理了稍多的内容，这些内容主要是 SeqMan 组合所带来的附加属性或附加信息。

首先是管理词典本身，即 benchmark_set 即正样本与负样本两序列库总和所对应的词典本身，包括直接获取词典，获得词典大小，词典字串等方法，词典类型定义为 AmoidDict，序列库具体实现是 Kmer 词典类 KmerBasedDict，序列库负责提供生成词典的用户接口，这些都将在稍后介绍。

其次是记录当前这些 SeqMan 使用了哪一种编码器来编码，即记录一个 EncoderType 对象。

最后提供获得编码结果字串的接口，便于输出和打印。

Constructor Summary

Constructors	
Constructor	**Description**
SeqLib (SeqGetter pGetter, SeqGetter nGetter, SeqGetter iGetter)	

Method Summary

All Methods

Modifier and Type	Method	Description
AmoidDict<java.lang.String>	getDict()throws SeqLibNullDictException	获取序列库的词典
int	getDictSize()throws SeqLibNullDictException	获得词典大小
java.lang.String	getDictString()throws SeqLibNullDictException	获取词典字串
java.lang.String	getEncodeStringI()throws SeqLibNotEncodedException	获取独立样本编码字串
java.lang.String	getEncodeStringN()throws SeqLibNotEncodedException	获取负样本编码字串
java.lang.String	getEncodeStringP()throws SeqLibNotEncodedException	获取正样本编码字串
SeqMan[]	getSeqMan()	获取序列管理器
SeqMan[]	getSeqMan_BenchmarkSet()	获取 BenchmarkSet 的序列管理器数组
SeqMan[]	getSeqMan_IndependentSet()	获取 IndependentSet 的序列管理器数组

Modifier and Type	Method	Description
void	initDict(int kmer_dim)	使用序列库进行编码前，必须先初始化词典
EncoderType	getEncodeType()	获取编码类型
void	setEncoderType (EncoderType t)	设定编码类型，这个是留给 EncoderHandler 调用的，编码完毕以后就设定一下序列库的编码类型

Package com.jbji.sample.excpetion

Class SeqLibNotEncodedException

Class SeqLibNullDictException

两个类都是为 SeqLib 而设计的异常类，目的是为了告诉用户，当前在没有出实现词典的情况下就进行了词典相关非法操作，或者在没有进行序列库编码的情况下就试图获取编码结果，这都将直接抛出异常，并且用户完全有能力处理这些异常。

II. 向量化模块（编码模块）

输入类型	未内嵌编码结果的序列库 SeqLib 对象，用于指定编码方式的 EncoderHandler
输出类型	内嵌编码结果的序列库 SeqLib 对象

包含的 package：com.jbji.Encoder com.jbji.Encoder.EncodeUtility com.jbji.Encoder.EncodeUtility.dict

向量化有两步，生成词典和进行编码，这两步都离不开从 SeqLib 获取 SeqMan 序列管理器。先说生成词典，生成词典由词典类负责，先来介绍词典类。

Class AmoidDict<T>

java.lang.Object
com.jbji.Encoder.EncodeUtility.dict.AmoidDict<T>

All Implemented Interfaces:

`java.lang.Iterable<T>`

Direct Known Subclasses:

`KmerBasedDict`

```
public abstract class AmoidDict<T>
extends java.lang.Object
implements java.lang.Iterable<T>
```

氨基酸组合词典 AmoidDict，一个典型的抽象类，目的是为了描述抽象的，不确定的词典，为更多词典方式的扩展提供进一步的接口。

Constructor Summary

Constructors	
Constructor	**Description**
`AmoidDict()`	

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
`java.util.Iterator<T>`	`iterator()`	
`int`	`size()`	

Class KmerBasedDict

java.lang.Object
com.jbji.Encoder.EncodeUtility.dict.AmoidDict<java.lang.String>
com.jbji.Encoder.EncodeUtility.dict.KmerBasedDict

All Implemented Interfaces:

`java.lang.Iterable<java.lang.String>`

```
public class KmerBasedDict
extends AmoidDict<java.lang.String>
```

Kmer 分词词典，本文最重要的分词词典，是抽象词典的具体实现。可以通过迭代器的访问方式来访问词典中的具体内容，后文编码器检索遍历整个词典时，正是采用了迭代器的方式。

Constructor Summary

Constructors	
Constructor	**Description**
KmerBasedDict(int dim)	初始化一个 Kmer 词典
KmerBasedDict (int dim, SeqMan... s)	按序列库建立一个 Kmer 词典，给定不定长度自变量，方便控制要用于建立序列库的 SeqMan 序列管理器

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
java.lang.String	toString()	

Methods inherited from class com.jbji.Encoder.EncodeUtility.dict.AmoidDict

iterator, size

至此，我们已经可以根据序列库来建立词典了，具体方式就是调用 SeqLib 的 initDict()方法，KmerBasedDict 对象就会自动建立。

下面介绍编码器。

Class EncoderHandler

java.lang.Object
com.jbji.Encoder.EncodeUtility.EncoderHandler

Direct Known Subclasses:

OneHotEncodingHandler, ResetHandler, TFIDFHandler

```
public abstract class EncoderHandler
extends java.lang.Object
```

-

编码处理器 EncoderHandler，是向量化的操作者，也是实现 Encoder 多态的一个抽象类，可以通过 EncoderHandler 的具体类实现来对 SeqLib 进行编码。

获得编码器类型是为了便于用户在保存文件或者记录日志时记录当前操作的类型。

Method Summary

All MethodsInstance MethodsAbstract MethodsConcrete Methods

Modifier and Type	Method	Description
abstract void	**encode**(**SeqLib** lib) throws SeqLibNullDictException	编码方法，操作这个方法就可以对序列库 SeqLib 编码，前提是已经生成了词典
EncoderType	**getType**()	获得当前编码器的类型

Enum EncoderType

用于记录编码器类型的枚举量

Enum Constants	
Enum Constant	**Description**
BAG_OF_WORDS	Bag Of Words 编码
IDF	逆文档频率频率编码（未使用）
ONE_HOT_ENCODING	One Hot Encoding 编码
TF	TermFrequency 词频（未使用）
TF_IDF	TF-IDF 编码
UNDEFINED	未定义编码

有了编码器类型，如果有需要，用户就可以通过 EncoderHandler 的容器来保存各种各样的操作模式，同时还能通过 getType()来得知具体是什么操作模式。

Class ResetHandler

```
public class ResetHandler
extends EncoderHandler
```

EncoderHandler 的子类，重置编码结果的编码器，用于调用编码器前清空已经存在的编码结果。

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
void	**encode**(**SeqLib** lib)	

Class OneHotEncodingHandler

java.lang.Object
com.jbji.Encoder.EncodeUtility.EncoderHandler
com.jbji.Encoder.OneHotEncodingHandler

Direct Known Subclasses:

BagOfWordsHandler

```
public class OneHotEncodingHandler
extends EncoderHandler
```

EncoderHandler 的子类，用于处理独热编码，由于与 BagOfWords 唯一的区别就是独热编码只关心对于给定序列单元，某个词典项是否出现，而不关心出现次数，因此我们不妨额外定义 count 函数，用于检查字符串字串，这样 Bag Of Words 只需要继承 OneHotEncodingHandler，简单重写 count 方法改变子串检查方式即可。

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
int	**count**(java.lang.String parent, java.lang.String child)	

Modifier and Type	Method	Description
void	encode(SeqLib lib)	

Class BagOfWordsHandler

java.lang.Object
com.jbji.Encoder.EncodeUtility.EncoderHandler
com.jbji.Encoder.OneHotEncodingHandler
com.jbji.Encoder.BagOfWordsHandler

```
public class BagOfWordsHandler
extends OneHotEncodingHandler
```

继承 OneHotEncodingHandler，重写 count 方法，实现 BagOfWords 编码器。

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
int	count(java.lang.String parent, java.lang.String child)	

Class TFIDFHandler

java.lang.Object
com.jbji.Encoder.EncodeUtility.EncoderHandler
com.jbji.Encoder.TFIDFHandler

```
public class TFIDFHandler
extends EncoderHandler
```

TFIDFHandler，处理 TFIDF 编码，调用方式与前两种没有区别，相比于前两种编码方式，这个实现起来还缺少词典的逆文档频率，因此我们定义了一个额外的类 DictIDFGenerator

Method Summary

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method	Description
void	encode(SeqLib lib)	

Class DictIDFGenerator

java.lang.Object
com.jbji.Encoder.EncodeUtility.DictIDFGenerator

All Implemented Interfaces:

java.lang.Iterable<java.lang.Float>

逆文档频率生成器的使用方法同样很简单，只需要给定序列管理器和词典即可，TFIDFHandler 正是通过调用该类生成对象，再调用迭代器去获得 IDF，进而计算 TFIDF 的。内部利用了三个 private 方法 initSeqs(),addItemAppearance(),calcIDF()来实现 IDF 的具体计算，具体计算细节不再详述。

Constructors	
Constructor	**Description**
DictIDFGenerator (AmoidDict<java.lang.String> dict, SeqMan... seqs)	

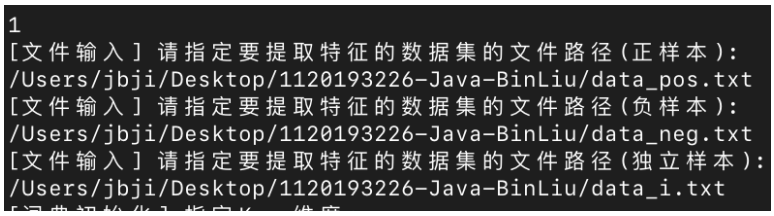
至此，只需要调用任意一种 EncoderHandler，利用 encode 方法去编码一个初始化好 dict 的 SeqLib 即可生成向量化编码结果，再通过 SeqLib 提供的相关接口就可以将结果获取，即可输出或利用 FileSaver 将其保存到文件。

3. 实验结果与分析

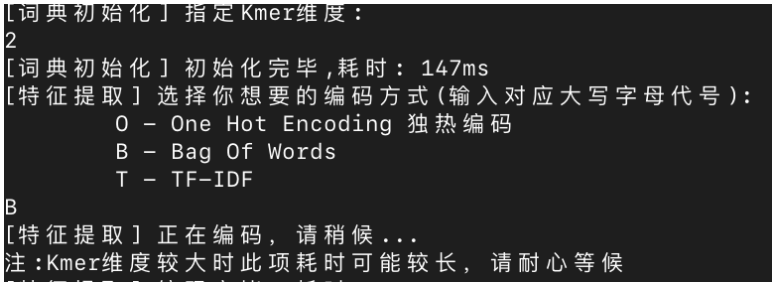
模型能够基于蛋白质序列文件输出整型或浮点型特征向量，程序提供了两种调用方式，命令行窗口和图形化界面。

命令行窗口

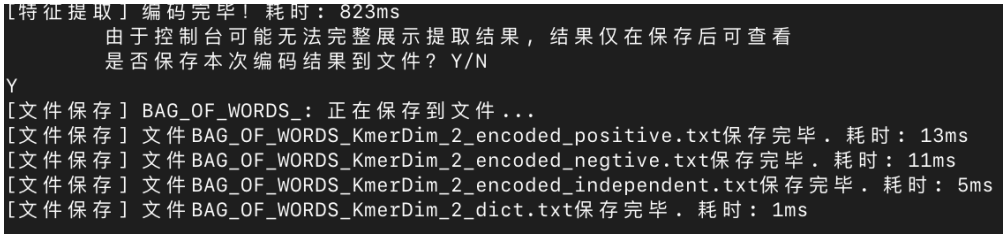
如果使用命令行启动程序，那么在弹出图形窗口的同时，还会出现命令行终端界面，我们输入任意字符进入程序流程，并根据提示指定文件输入，终端提示如下。



指定 Kmer 维度并选择编码方式终端反馈如下，



处理完毕的界面如下



选择 Y 即可将文件全部保存，终端重新进入初始流程。

图形化界面

程序的整体运行效果如图 3.1、图 3.2 所示

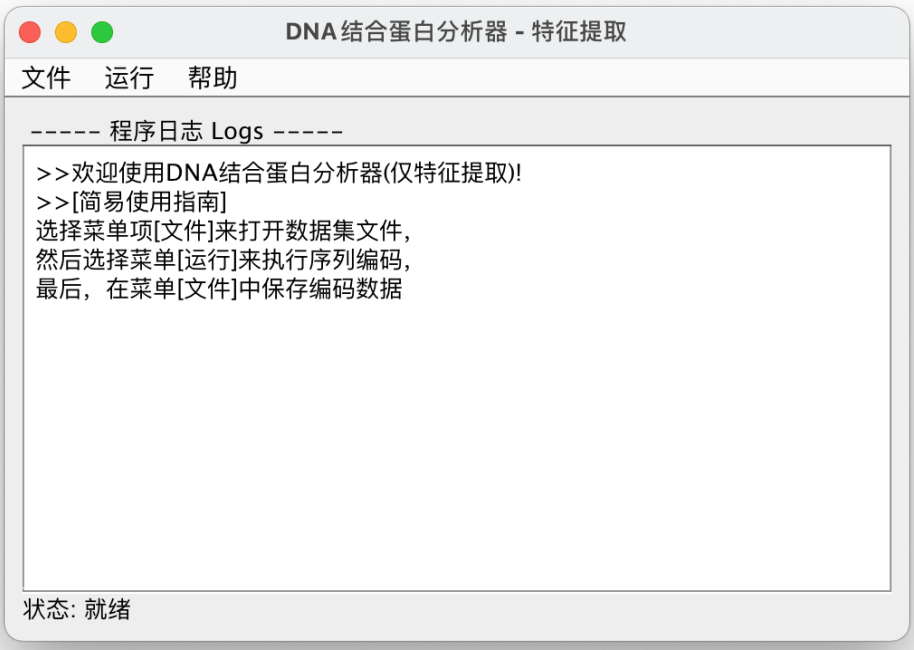


图 3.1 程序在 MacOS 上的运行效果（OpenJDK 15.0.1）

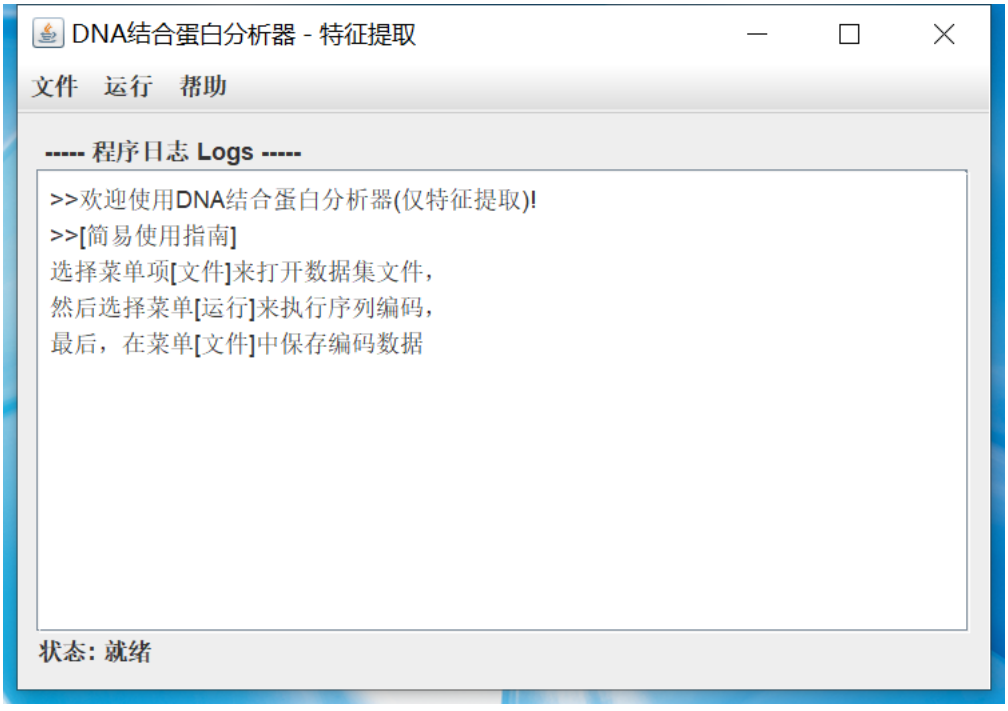


图 3.2 程序在 Windows 10 上的运行效果（Java 15）

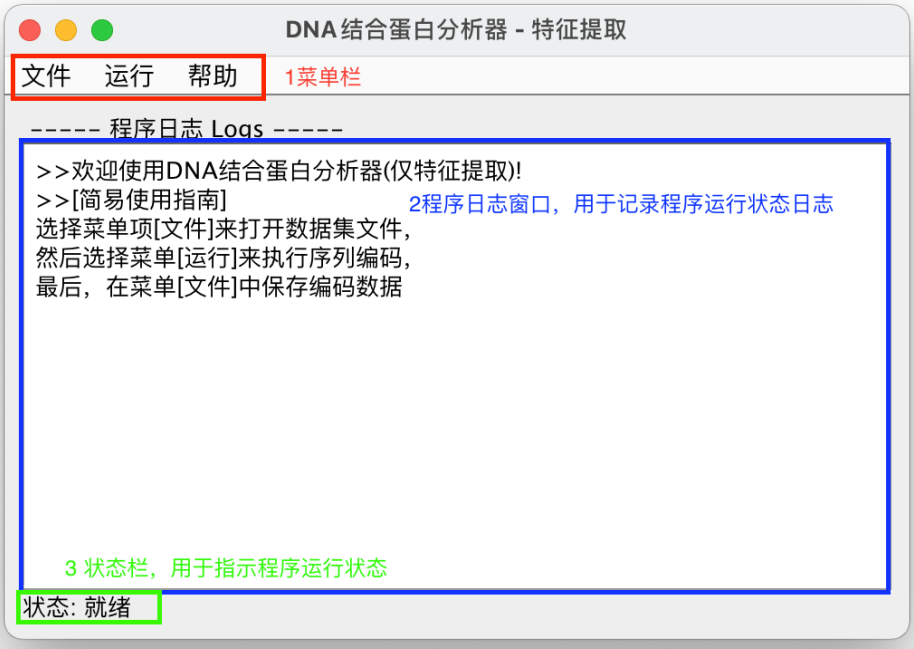


图 3.3 程序窗体的主要控件

程序窗体的主要控件如图 3.3 所示，包括菜单栏、日志、以及状态栏三部分。

菜单栏包括文件、运行和帮助三个菜单。日志窗口不可编辑。

点击文件菜单，包含的菜单项如图 3.4 所示，其中包括打开和保存两大类菜单项。

图 3.4 菜单项

在没有进行编码的情况下直接进行保存文件，程序会直接提示错误，并将其记录在日志中。

图 3.5 错误窗口

>>错误：特征提取未进行，未生成编码结果，无法保存至文件

图 3.6 错误日志

现在我们打开三种样本数据集，点击打开菜单项，将弹出文件选择窗口

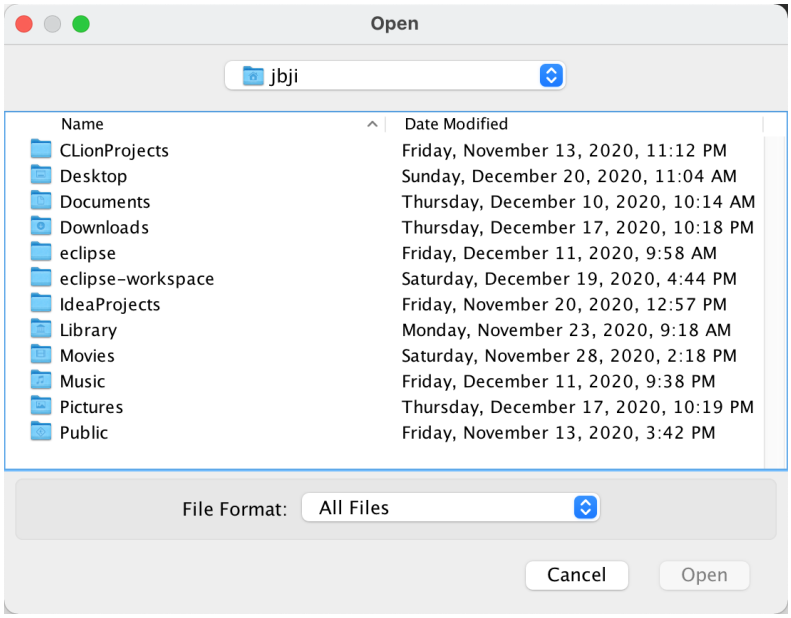


图 3.7 文件选择窗口

选择文件，点击打开，将产生对应日志

图 3.8 选择文件产生的日志

如果打开的文件数量不足，点击运行会产生错误

图 3.9 数据集不足产生的错误

```
>>错误：数据集文件不足，请先选择三个数据集文件.
```

图 3.10 数据集不足产生错误的日志

打开全部三个数据集后，点选运行菜单

图 3.11 运行菜单

这里可以选择任意一种编码方式，我们选择 TF-IDF 编码，此时程序会要求你输入 Kmer 中分词的大小，同时日志窗口显示正在处理 TF-IDF，状态栏显示正在处理。

图 3.12 选择 TF-IDF 后的窗口状态

如果此时的输入格式不正确，会产生相应提示，同时日志栏提示操作取消。

图 3.13 3.14，Kmer 输入无效的错误提示

我们输入一个有效的 Kmer，比如 2，点击确定后程序会自动进行处理，并记录操作相关信息。

图 3.15 一个成功的操作

此时点击菜单栏中文件中的保存方式即可保存数据，这里选择保存全部数据至当前文件夹，此时保存状态会在日志中被记录

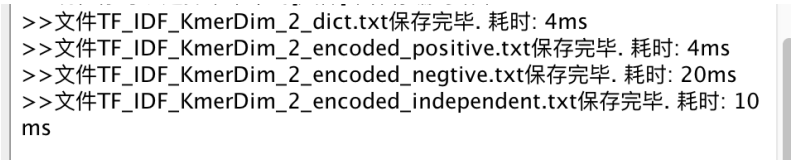


图 3.16 文件保存的日志记录

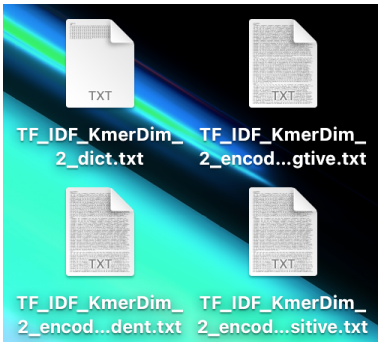


图 3.17 保存得到的文件

文件内容如图 3.18，3.19 所示

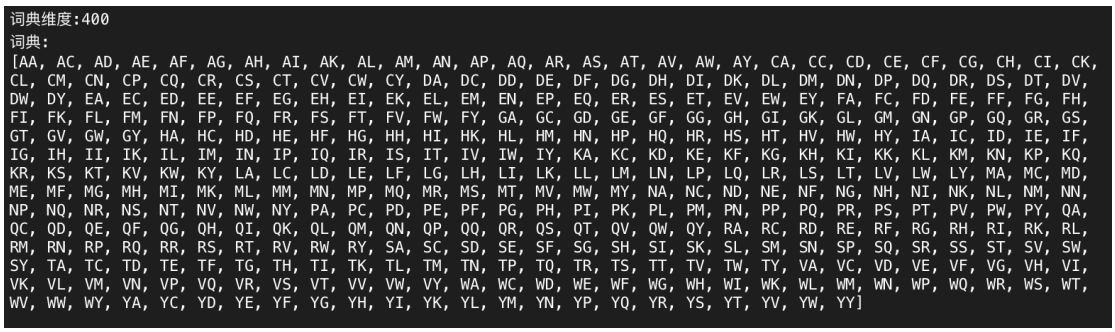


图 3.18 3.19 保存的文件内容

此时可以点击运行以编码方式继续生成向量化编码结果，或者打开其他文件进行相同操作。

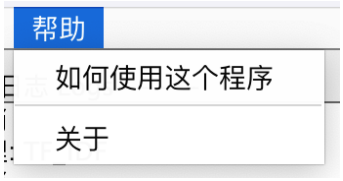


图 3.20 帮助菜单

帮助菜单可以提供有限的帮助，点击如何使用这个程序将在日志栏生成简易使用指南（图 3.21），点击关于则会报告作者信息（图 3.22）。

图 3.21，3.22 帮助菜单与关于窗口

附录：开发文档

1. 开发环境
   JavaSE-1.8 (Amazon Corretto 8 [1.8.0_275])
2. 开发平台
   Eclipse IDE for Java Developers (includes Incubating components)
   Version: 2020-09 (4.17.0) Build id: 20200910-1200
3. 项目内容
   可执行程序：DNA-Protein.jar
   数据集示例文件：data_pos.txt data_neg.txt data_i.txt
   数据集在 Kmer=2 时的 TFIDF 示例输出：Example_TFIDF_Kmer2.zip
   程序源代码（包含 java-doc）：DNA-Protein
4. 程序调用方法
   1) 安装版本不低于 JRE 8（推荐）的 Java 运行环境
   2) 将程序jar文件（DNA-Protein）复制到计算机上，准备好样本数据集文件，或者使用程序给出的示例文件
   3) 如果要同时调用命令行界面和图形界面，则
      在项目所在目录进入cmd或powershell 或 terminal界面，通过java -jar DNA-protein.jar 来运行程序。
   4) 如果只希望调用图形界面，则在设置好文件关联的前提下直接打开 DNA-Protein.jar 文件，或者选择以 Java Runtime Environment 打开。
   5) 对于命令行界面的调用
      a) 输入任意字符回车，进入程序流程
      b) 程序会要求指定样本数据集路径，通过拖拽文件或者直接输入文件名的方式，依次指定正样本、负样本、独立样本文件。
      c) 之后程序会要求指定 Kmer 维度，这是为了建立词典，建议输入不大于 4 的正整数
      d) 此后程序给出编码方式选择，提供 O - OneHotEncoding 编码，B - BagOfWords 编码，T - TF-IDF 编码三种方式可供选择
      e) 编码完毕，会询问你是否需要将其保存到文件，输入 Y 会将向量化编码结果保存到当前目录。

6) 对于图形界面的调用
   a) 点击菜单：文件-打开正数据集，选择正数据集文件
   b) 点击菜单：文件-打开负数据集，打开负数据集文件
   c) 点击菜单：文件-打开独立数据集，打开独立数据集文件
   d) 点击运行 - 选择一种编码方式，如果之前操作都正确，会弹出输入窗口
   e) 在新弹出的窗口中输入 Kmer 维度（建议输入不大于 4 的正整数），点击确定
   f) 程序将进行处理，并在程序日志区域显示运行时间，Kmer 维度较大或样本集较大时，处理速度可能较慢，请耐心等候。
   g) 点击菜单：文件 - 保存全部数据到当前文件夹，程序会自动保存运行结果到可执行 jar 包所在的文件夹，选择其他保存选项可以保存部分结果。
   h) 你可以选择其他文件或者更改向量化编码运行方式来执行程序和保存文件。