Министерство образования Республики Беларусь
Белорусский государственный университет
Биологический факультет
Кафедра клеточной биологии и биоинженерии растений

СОГЛАСОВАНО
Заведующий кафедрой
_____ Яковец О.Г.
«18» апреля 2024 г.

СОГЛАСОВАНО
Декан факультета
_____ Демидчик В.В.
«26» апреля 2024 г.

Программирование на Python в биологии

Электронный учебно-методический комплекс для специальности
6-05-0511-05 «Биоинженерия и биоинформатика»

Регистрационный № 2.4.2-24/467

Автор:
Бондаренко В. Ю., старший преподаватель

Рассмотрено и утверждено на заседании Научно-методического совета БГУ
30.04.2024 г., протокол № 7.

Минск 2024

УДК 57:004.032.045(075.8)
Б 811

Утверждено на заседании Научно-методического совета БГУ
Протокол № 7 от 30.04.2024 г.

Решение о депонировании вынес:
Совет биологического факультета
Протокол № 8 от 26.04.2024 г.

А в т о р :
Бондаренко Владислав Юрьевич, старший преподаватель кафедры клеточной биологии и биоинженерии растений биологического факультета БГУ

Рецензенты:
Шашко Ю. К., директор Института почвоведения и агрохимии НАН Беларуси, д. с.-х. н., профессор;
Недзьведь А. М., заведующий кафедрой информационных систем управления ФПМИ БГУ, д.т.н., доцент.

Бондаренко, В. Ю. Программирование на Python в биологии : электронный учебно-методический комплекс для специальности: 6-05-0511-05 «Биоинженерия и биоинформатика» / В. Ю. Бондаренко ; БГУ, Биологический фак., Каф. клеточной биологии и биоинженерии растений. – Минск : БГУ, 2024. – 68 с. : ил. – Библиогр.: с. 67–68.

Электронный учебно-методический комплекс (ЭУМК) предназначен для студентов I ступени высшего образования специальности: 6-05-0511-05 «Биоинженерия и биоинформатика». Содержание ЭУМК посвящено формированию у студентов целостной системы знаний по применению языка программирования Python для решения задач биологии и биоинформатики.

СОДЕРЖАНИЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Настоящий электронный учебно-методический комплекс (ЭУМК) включает материал для проведения занятий по учебной дисциплине «Программирование на Python в биологии» для специальности 6-05-0511-05 «Биоинженерия и биоинформатика». ЭУМК создан в соответствии с учебной программой УД-11733/уч, на основе ОСВО 1-31 01 04-2021 и учебного плана УВО № G 31-1-206/уч.

Учебная дисциплина « Программирование на Python в биологии» относится к компоненту учреждения высшего образования учебного плана. Цель учебной дисциплины – сформировать у студентов целостную систему знаний по применению языка программирования Python для решения задач биологии и биоинформатики.

Целью ЭУМК является предоставление теоретической и методологической базы, а также других сведений, необходимых для успешного освоения учебной дисциплины « Программирование на Python в биологии». ЭУМК охватывает теоретические основы и содержит описание методик по тематическим разделам: «Алгоритмы для анализа геномных данных», «Анализ протеомных данных», «Машинное обучение и анализ изображений».

В структуру ЭУМК входит:

1) теоретический раздел (содержит краткий конспект лекций);

2) практический раздел (включает задания для самостоятельного выполнения в рамках подготовки к практическим занятиям, а также тематику практических работ);

3) контроль самостоятельной работы студентов (содержит примерный перечень заданий для управляемой самостоятельной работы обучающихся, темы рефератов, примерный перечень вопросов к зачету);

4) вспомогательный раздел (содержит списки основной и дополнительной литературы).

Работа с ЭУМК должна включать на первом этапе ознакомление с тематическим планом учебной дисциплины, представленным в учебной программе УВО. Информацию о тематике лекций и практических занятий, перечнях рассматриваемых вопросов и рекомендуемой для их изучения литературы можно получить из рабочего варианта учебной программы по учебной дисциплине. Для подготовки к практическим занятиям рекомендуется использовать материалы, представленные в теоретическом разделе ЭУМК, а также материалы для текущего контроля самостоятельной работы. В ходе подготовки к зачету целесообразно ознакомиться с требованиями к компетенциям по учебной дисциплине, изложенными в учебной программе УВО, а также перечнем вопросов к зачету, приведенным в настоящем ЭУМК. Для написания рефератов могут быть использованы информационно-аналитические материалы, указанные в соответствующем разделе ЭУМК.

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Предмет биоинформатики

Вопросы:
1. Предмет, цели и задачи биоинформатики.
2. Использование языков программирования в биологии.
3. Классы в Python.
4. Задачи геномики.

Биоинформатика – это междисциплинарная область, которая объединяет общую биологию, молекулярную биологию, геномику, компьютерные науки, статистику и технологии обработки данных для анализа биологических данных. Биоинформатика используется для изучения и понимания биологических систем, таких как геномы, протеомы и метаболомы. Она включает в себя изучение и разработку компьютерных методов для получения, анализа, хранения, организации и визуализации биологических данных. Биоинформатика в широком смысле подразумевает работу с любыми видами биологических данных, например, электронных микрофотографий или целых геномов организмов.

Цель биоинформатики заключается в применении методов анализа данных и информационных технологий для изучения биологических систем. Основные задачи биоинформатики включают:

1. Анализ генетических данных: Исследование геномов, поиск вариаций в генетических последовательностях, анализ экспрессии генов и работа с биологическими последовательностями.
2. Предсказание структуры белков: Разработка методов для предсказания трехмерной структуры белков на основе их аминокислотной последовательности.
3. Изучение взаимодействий биомолекул: Анализ взаимодействий между белками, нуклеиновыми кислотами и другими биологическими молекулами.
4. Функциональная аннотация геномов: Определение функций неизвестных генов и генетических элементов на основе сравнительного анализа и экспериментальных данных.
5. Анализ биологических изображений: Разделение изображений на отдельные области для выделения интересующих объектов. Идентификация и классификация биологических объектов на изображениях. Обнаружение и диагностика патологических изменений на изображениях, таких как опухоли, воспаления или деформации тканей.
6. Разработка биоинформационных инструментов: Создание программного обеспечения и методов для анализа биологических данных и моделирования биологических процессов.

Целью всех этих задач является получение новых знаний о биологических системах, разработку лекарственных препаратов, диагностики заболеваний и осуществление персонализированной медицины.

Несомненно, для использования всех возможностей биоинформатики необходимы инструменты, их можно условно разделить на user-friendly и user-unfriendly. К первой категории относятся различные компьютерные программы, разработанные специально для решения конкретных биологических задач, в таких инструментах пользователь вносит данные, выбирает тип обработки/модификации и получает результат в виде таблиц или графиков. В таких программах есть один паттерн поведения, который заложен в коде, и зачастую нет разнообразия поведений, который необходим пользователю. Например, программы для анализа результатов полногеномного секвенирования организмов, в которых отсутствуют настройки поиска генов токсинов, эндолизинов, деполимераз при работе с бактериофагами. Инструмент, очевидно, полезный, но иногда его недостаточно. В таких случаях стоит рассмотреть способы создания своих собственных программ и для этого используют тот самый user-unfriendly инструмент, однако, таковым он является только в самом начале освоения. Языки программирования широко используются в биологических науках и биоинформатике. Они позволяют создавать свои собственные варианты развития событий, учитывать мелочи, о которых обычный пользователь не задумывается, видеть алгоритм, а не просто его использовать, создавать что-то новое, а не пользоваться готовым. Среди популярных и высокоэффективных языков стоит отметь Python.

Python – является популярным выбором для биоинформатики благодаря своей простоте, богатству на библиотеки и широкому спектру инструментов для научных вычислений. Он часто используется для анализа данных геномики, секвенирования ДНК, визуализации биологических данных и машинного обучения.

Перед началом использования Python стоит помнить о некоторых рекомендациях, одной из которых является – виртуальная среда (virtual environments). Виртуальная среда Python представляет собой инструмент, который позволяет создавать изолированные окружения для управления зависимостями и пакетами Python в проекте. Это позволяет изолировать различные проекты, чтобы у каждого из них были свои собственные версии библиотек Python, предотвращая конфликты между различными версиями и зависимостями.

Виртуальные среды полезны, когда вам нужно работать с несколькими проектами, каждый из которых требует определенных версий библиотек или зависимостей. Они также удобны при работе в командной строке, так как вам не нужны административные права для установки пакетов, и они помогают держать вашу установку Python чистой и упорядоченной.

Для создания виртуальной среды в Python часто используется инструмент venv или virtualenv. Здесь и далее будет представлены команды в командной

строке или терминале. Вы можете создать новую виртуальную среду с помощью следующей команды:

python -m venv имя_вашей_среды

После создания виртуальной среды, вы можете активировать ее и устанавливать в нее требуемые библиотеки и зависимости, которые будут доступны только в пределах этой среды:

Windows:
имя_вашей_среды\Scripts\activate
pip install pandas

Unix or MacOS:
. имя_вашей_среды/bin/activate
pip install pandas

Чтобы выйти из активированной виртуальной среды Python, вы можете использовать одну из двух команд, в зависимости от операционной системы:

deactivate

Команда deactivate деактивирует виртуальную среду, возвращая в базовую системную установку Python.

Немаловажным являются зависимости каждого проекта. Файл requirements.txt в Python используется для управления зависимостями вашего проекта. В этом файле приводятся список всех библиотек и их версий, которые используются в вашем проекте.

Этот файл полезен для следующих целей:

1. Воспроизводимость: Предоставляет возможность точного воссоздания окружения для других разработчиков, работающих над вашим проектом. Каждый, кто будет развивать проект, может использовать этот файл для установки тех же версий библиотек, что и у вас. Это обеспечивает единое окружение разработки для всей команды.
2. Управление зависимостями: Позволяет легко устанавливать все необходимые библиотеки и их версии с помощью одной команды, вместо того чтобы устанавливать каждый пакет вручную.
3. Сборка и развертывание: При развертывании проекта на другом компьютере или сервере, можно использовать файл requirements.txt для установки всех необходимых зависимостей одной командой. Таким образом, это упрощает процесс развертывания проекта.

Чтобы создать файл requirements.txt, вы можете использовать команду:

pip freeze > requirements.txt

Это запишет список всех установленных библиотек и их версий в файл requirements.txt.

При работе с виртуальными средами Python, файл requirements.txt используется для установки необходимых пакетов в виртуальную среду, обеспечивая воспроизводимость окружения и управление зависимостями.

pip install -r requirements.txt

Хотя файл requirements.txt полезен для управления зависимостями в проекте, у него также есть некоторые проблемы и ограничения:

1. Версионные конфликты: Иногда бывает сложно управлять версиями зависимостей. Если вы явно указываете версии библиотек в файле requirements.txt, то может возникнуть ситуация, когда одна библиотека требует определенной версии другой, и это может привести к версионным конфликтам.
2. Дублирование зависимостей: Если вы работаете над несколькими проектами, каждый из которых имеет свой собственный файл requirements.txt, то ваши зависимости могут дублироваться между проектами. Это может привести к избыточной использованной памяти и хранению.
3. Фиксированные версии: Иногда фиксирование версий всех зависимостей в файле requirements.txt может создать дополнительную работу при обновлении зависимостей. Если вы часто обновляете свои библиотеки, вам может приходиться регулярно обновлять версии в файле requirements.txt.
4. Отслеживание изменений: Когда изменяются зависимости, требуется обновлять файл requirements.txt вручную, что может быть подвержено ошибкам.

Для решения этих проблем существуют другие инструменты и методы управления зависимостями, включая использование менеджеров зависимостей, таких как Pipenv, Poetry или использование файлов зависимостей в формате JSON или TOML, которые могут предложить более гибкий и удобный способ управления зависимостями в Python.

Следующим важным элементом, который нам необходимо затронуть – **классы**. В Python классы представляют собой основу объектно-ориентированного программирования (ООП). Класс определяет структуру для создания объектов, которые могут содержать атрибуты (переменные) и методы (функции). Среди основных моментов, которые стоит обозначить при создании класса следует выделить: метод __init__ и ключевое слово self.

Метод __init__ является специальным методом (методом "конструктором") в Python, который вызывается автоматически при создании нового экземпляра класса. Он используется для инициализации атрибутов объекта. Это значит, что при создании экземпляра класса, этому экземпляру будут присвоены его личные атрибуты (переменные). Метод __init__ не является обязательным для классов в Python, но его использование позволяет создавать более надежные и

удобные классы, так как он позволяет задать начальные значения атрибутов объекта при его создании.

В Python ключевое слово self используется в определениях методов класса для ссылки на экземпляр этого класса. self используется в методах, чтобы обозначить, что методы работают с атрибутами экземпляра. Пример использование метода __init__ и ключевого слова self:

```
class Snake:
    def __init__(self, name):
        self.name = name

    def voice(self):
        return 'some_shhh'

s1 = Snake('snake1')
```

Когда мы создаем экземпляр класса s1 = Snake('snake1'), self указывает на этот конкретный экземпляр (в данном случае s1), что позволяет методам получать доступ к его атрибутам. При вызове функции voice и атрибута экземпляра name будет выведено следующее:

```
print(s1.voice(), s1.name)
#some_shhh snake1
```

Ещё один базовый пример определения класса в Python:

```
class Snake:
    name = 'Python'

    def __init__(self):
        self.mouse_eaten_counter = 0

    def voice(self):
        self. mouse_eaten_counter += 1
        return 'shhh'

s1 = Snake()
s2 = Snake()
```

Этот пример описывает класс Snake, который имеет атрибуты класса name, атрибут экземпляра mouse_eaten_counter, метод __init__, который используется для инициализации атрибутов при создании экземпляра, и метод voice, который возвращает строку.

Когда создается экземпляр класса (s1 = Snake()), экземпляру присваивается атрибут класса name. Далее при вызове метода voice у конкретного экземпляра класса происходит увеличение переменной (инкремент) mouse_eaten_counter на единицу и возвращение строку 'shhh'.

```
print(s1.voice(), s1.mouse_eaten_counter, s2.mouse_eaten_counter)
#shhh 1 0
```

Python также поддерживает наследование, множественное наследование и другие возможности ООП. Классы могут использоваться для создания сложных структур данных, моделирования объектов реального мира и многого другого. ООП позволяет создавать более модульный и масштабируемый код, что делает его удобным для решения различных задач программирования.

Python наследование позволяет создавать новый класс на основе уже существующего класса. Новый класс, называемый подклассом или производным классом, наследует атрибуты и методы от родительского класса, который иногда называют суперклассом или базовым классом.

Для определения наследования в Python используется следующий синтаксис:

```
class Reptilia:
    def reptilia_method(self):
        return 'Это родительский метод из класса Reptilia'

class Snake(Reptilia):
    def snake_method(self):
        return 'Это метод из дочернего класса'

s1 = Snake()

print(s1.snake_method(), '\n', s1.reptilia_method())
# Это метод из дочернего класса
#Это родительский метод из класса Reptilia
```

В этом примере Snake наследует функционал класса Reptilia. Это означает, что Snake может использовать все атрибуты и методы, определенные в Reptilia, а также иметь свои собственные атрибуты и методы.

Наследование позволяет создавать иерархии классов, где базовые функциональности определены в родительских классах, а более конкретные или специфические функции могут быть добавлены в дочерних классах. Это способствует повторному использованию кода, упрощает его поддержку и обеспечивает более надежную структуру программы.

Python также поддерживает множественное наследование, когда дочерний класс наследует функционал от нескольких родительских классов. В этом случае синтаксис будет выглядеть так:

```
class Animals:
    def animals_method(self):
        return 'Это родительский метод из класса Animals'

class Reptilia:
    def reptilia_method(self):
        return 'Это родительский метод из класса Reptilia'

class Snake(Animals, Reptilia):
    def snake_method(self):
        return 'Это метод из дочернего класса'
```

Однако использование множественного наследования требует осторожности, и в большинстве случаев рекомендуется использовать одиночное наследование для обеспечения чистого и понятного проектирования кода.

Полиморфизм в объектно-ориентированном программировании, в том числе и в Python, относится к способности объектов различных классов обеспечивать унифицированный интерфейс для обработки различных типов данных. Это означает, что объекты различных классов могут обладать одинаковым интерфейсом (например, иметь методы с одинаковыми именами), но предоставлять различную реализацию для этих методов, в зависимости от своего типа или класса.

Полиморфизм в Python может быть реализован с помощью методов с одинаковыми именами, но принимающими различное количество аргументов. Часто это достигается с использованием методов перегрузки операторов. Пример полиморфизма в Python:

```
class Animals:
    def method(self):
        return 'Это родительский метод из класса Animals'

class Reptilia(Animals):
    def method(self):
        return 'Это родительский метод из класса Reptilia'

class Snake(Reptilia):
    def method(self):
        return 'Это метод из дочернего класса'
```

```
s1, s2, s3 = Snake(), Reptilia(), Animals()


print(s1.method(), s2.method(), s3.method(), sep='\n')
#Это метод из дочернего класса
#Это родительский метод из класса Reptilia
#Это родительский метод из класса Animals
```

В этом примере у нас есть классы Animals, Reptilia и Snake, у которых есть метод *method*. В результате несмотря на то, что каждый экземпляр имеет одно и то же имя метода, обеспечивается различная реализация в зависимости от типа объекта, что и является проявлением полиморфизма.

Инкапсуляция в Python – это механизм, который позволяет скрыть детали реализации класса от пользователя, ограничив доступ к атрибутам и методам класса. Это достигается путем объявления атрибутов и методов класса как приватных или защищенных.

Атрибуты и методы, начинающиеся с двойного подчеркивания (__), считаются приватными и не могут быть доступны за пределами класса, где они были объявлены. Однако, они могут быть доступны внутри класса, в котором они были объявлены.

```
class NewClass:
    def __init__(self):
        self.__private_attr = 10

    def __private_method(self):
        return 'This is a private method'

obj = NewClass()

print(obj.__private_attr)
# AttributeError: 'NewClass' object has no attribute '__private_attr'
print(obj.__private_method())
# AttributeError: 'NewClass' object has no attribute '__private_method'
```

Атрибуты и методы, начинающиеся с одного подчеркивания (_), считаются защищенными. Это предупреждение для других программистов о том, что эти атрибуты и методы не должны использоваться вне класса, но это не запрещает их использование.

```
class NewClass:
    def __init__(self):
        self._protected_attr = 20
```

```
    def _protected_method(self):
        return 'This is a protected method'

obj = NewClass()
print(obj._protected_attr)
#20
print(obj._protected_method())
#This is a protected method
```

Использование инкапсуляции помогает создавать более безопасные и надежные программы, так как скрывает сложность реализации и предоставляет более четкий интерфейс для взаимодействия с классом. Пример:

```
class Snake:

    def __init__(self):
        self.__mouse_eaten_counter = 0

    def counter(self):
        return f'Количество поетых мышей: {self.__mouse_eaten_counter}'

    def change_counter(self, num):
        self.__mouse_eaten_counter = num

s1 = Snake()
print(s1.counter())
# Количество поетых мышей: 0

#запрет на изменение приватной переменной
s1.__mouse_eaten_counter = 12
print(s1.counter())
# Количество поетых мышей: 0

#функция изменения каунтера
s1.change_counter(12)
print(s1.counter())
# Количество поетых мышей: 12
```

Структура ДНК, РНК. Представляя работу биоинформатика, многие думают о работе с геномными или протеомными последовательностями, по факту всё так, хоть и такое представление ограниченное. Самым популярным объектом в биоинформатике действительно являются геномные последовательности. Кратко вспомним, что это такое и из чего они состоят.

Нуклеиновые кислоты (ДНК, РНК) построены из нуклеотидных звеньев, которые в свою очередь состоят из азотистого основания, углеводного остатка и фосфатной группы. Азотистые основания – это ароматические гетероциклические соединения, производные пиримидина или пурина. Имеется всего пять соединений этого класса, которые являются основными структурными компонентами нуклеиновых кислот. Соединения азотистых оснований с рибозой (РНК) или 2-дезоксирибозой (ДНК) носят название нуклеозиды (аденозин, гуанозин, уридин, тимидин, цитидин). Добавление фосфатной группы к нуклеозиду образует нуклеотид.

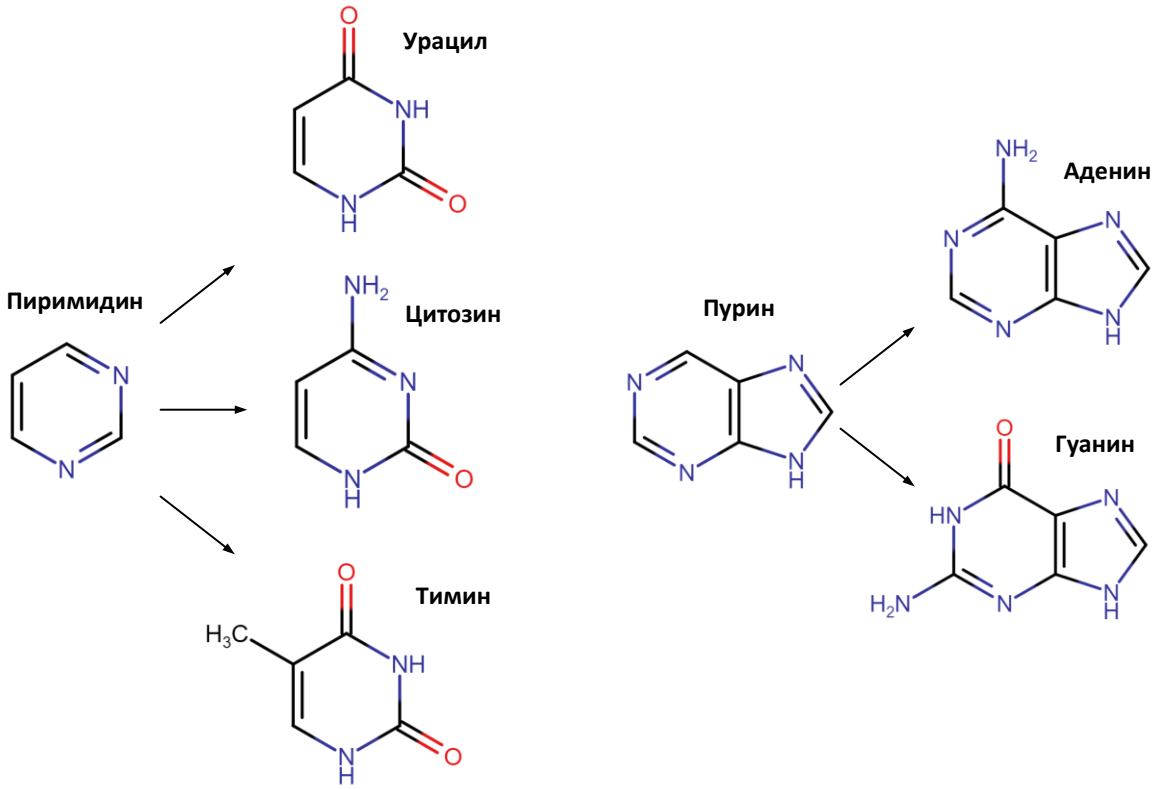


Рисунок 1 – Производные пурина и пиримидина (азотистые основания)

Комплементарность нуклеотидов – способность азотистых оснований спариваться специфично. Аденин формирует две водородные связи с тимином, а гуанин – три с цитозином.

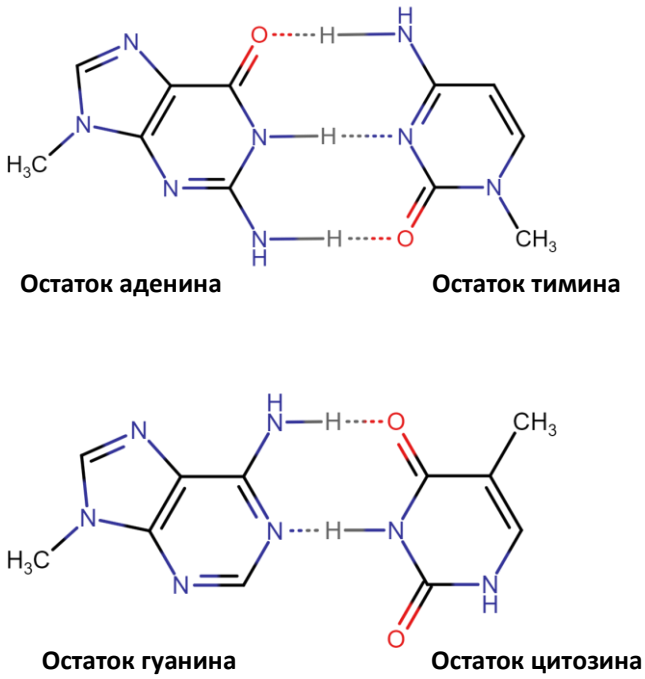


Рисунок 2 – Комплементарность азотистых оснований

Эти правила комплементарности нуклеотидов являются основой для образования двуцепочечной структуры ДНК и РНК, что позволяет точно копировать и передавать генетическую информацию в процессах репликации, транскрипции и трансляции.

Рассмотрим ключевые процессы, обеспечивающие передачу и экспрессию генетической информации в клетках:

1. Репликация является процессом копирования ДНК перед делением клетки. Происходит в ядрах клеток. ДНК разделяется на две цепи, и каждая из них служит в качестве матрицы для синтеза новой цепи, таким образом, образуется две идентичные молекулы ДНК. Результатом репликации являются две молекулы ДНК, каждая из которых состоит из одной старой и одной новой цепи.

2. Транскрипция представляет собой процесс синтеза РНК на основе ДНК матрицы. Происходит в ядрах клеток (в случае эукариот) или в цитоплазме (в случае прокариот). В процессе транскрипции, одна цепь ДНК служит в качестве матрицы для синтеза молекулы РНК. Полученная молекула РНК содержит ту же информацию, что и одна из цепей исходной ДНК.

3. Трансляция относится к процессу синтеза белков на основе информации, закодированной в молекулах РНК. Происходит в цитоплазме клеток, где рибосомы используют информацию с РНК для синтеза цепи аминокислот, которая затем складывается в белок. Рибосомы сканируют молекулу РНК, "читая" последовательность триплетов нуклеотидов (так называемых кодонов) и добавляя соответствующие аминокислоты.

Вместе эти процессы обеспечивают передачу и экспрессию генетической информации от ДНК к РНК, а затем к белкам, что является основой для всей

генетической детерминанты биологических организмов. Эти знания понадобятся нам для дальнейшей работы.

Геномика – это область науки, которая изучает геномы организмов, включая их структуру, функцию, эволюцию и взаимодействие с окружающей средой. Задачи геномики охватывают широкий спектр исследований и приложений, включая следующие:

1. Одна из основных задач геномики – это определение последовательности нуклеотидов в геноме организма. Это может быть выполнено с помощью различных методов секвенирования, таких как Sanger sequencing, Illumina sequencing, PacBio sequencing, и Nanopore sequencing.
2. После секвенирования генома необходимо собрать фрагменты последовательностей в исходную последовательность генома. Эта задача представляет собой сложную проблему из-за повторяющихся участков, ошибок секвенирования и других факторов.
3. После того, как геном собран, следующим шагом является аннотация, т.е. определение функций различных участков генома, таких как гены, регуляторные элементы, повторяющиеся последовательности и т.д.
4. Сравнительная геномика включает сравнение геномов разных организмов для выявления консервативных и изменчивых областей, а также идентификации генов, которые являются уникальными для определенных организмов или вида.
5. Эволюционная геномика изучает изменение геномов в процессе эволюции и выявляет факторы, которые влияют на эволюцию геномов, такие как мутации, рекомбинации, геномные дупликации и др.
6. Функциональная геномика изучает функции генов и их регуляторных элементов, а также их взаимодействие с другими биологическими системами, включая биохимические пути и клеточные процессы.
7. Метагеномика изучает геномы микроорганизмов, которые находятся в определенной среде, например, в почве, воде или кишечнике, с целью понять структуру и функцию микробных сообществ.
8. Геномика играет важную роль в развитии персонализированной медицины, которая основана на анализе индивидуальных геномов для предсказания риска заболеваний, выбора наиболее эффективного лечения и предотвращения побочных эффектов лекарств.

Эти и другие задачи геномики имеют важное значение для понимания биологических процессов, эволюции жизни на Земле и разработки новых методов диагностики и лечения заболеваний.

1.2. Применение Python для решения задач молекулярной биологии и геномики

Вопросы:
1. Основные компоненты ПЦР.
2. Возможности библиотеки BioPython.
3. Трансляция.

Для начала вспомним всем известный метод из молекулярной биологии и далее на его примере рассмотрим, чем нам может помочь Python. Полимеразная цепная реакция (ПЦР) – это метод, используемый для умножения определенного фрагмента ДНК в миллионы раз, что позволяет получить значительное количество копий данного фрагмента. ПЦР является важным инструментом в молекулярной биологии и имеет широкий спектр применений, включая молекулярную диагностику, исследования генома, клонирование генов и многие другие области.

Основные компоненты ПЦР:

1. Матрица ДНК – исходный образец ДНК, содержащий фрагмент, который необходимо умножить.
2. Праймеры – короткие одноцепочечные олигонуклеотиды (чаще от 15 до 30 нуклеотидов), которые специфически связываются с конкретной областью целевой ДНК и служат начальной точкой для синтеза новой ДНК.
3. ДНК-полимераза – фермент, который катализирует синтез новой ДНК вдоль матрицы, используя праймеры.
4. Десоксинуклеозидтрифосфаты (dNTPs) – Вещества, необходимые для синтеза новой ДНК, включая аденин (A), тимин (T), гуанин (G) и цитозин (C).

Стадии ПЦР:

1. Денатурация (разделение) – исходная двухцепочечная матричная ДНК нагревается (96°C), чтобы разделить две цепи, образуя одноцепочечные шаблоны.
2. Отжиг (присоединение праймеров) – реакционная смесь охлаждается, чтобы позволить праймерам специфически связаться с целевой областью ДНК.
3. Экстенсия (синтез новой ДНК) – ДНК-полимераза синтезирует новые цепи ДНК, используя праймеры в качестве начальной точки и dNTPs как строительные блоки.
4. Повторение цикла – процесс денатурации, отжига и экстенсии повторяется многократно, что приводит к экспоненциальному увеличению количества копий целевого фрагмента.

ПЦР является быстрым и эффективным методом умножения ДНК, который широко используется в исследованиях, медицинской диагностике, судебной экспертизе, а также в промышленности и сельском хозяйстве.

Несомненно, для вас эта хорошо известно, однако подготовка материала для вышеупомянутого метода тоже заслуживает внимания. Так какие из параметров стоит учитывать для проведения ПЦР. Длина праймера – использование слишком короткого праймера может привести к неспецифическому отжигу, тогда как создание слишком длинного праймер будет стоить дороже. Разработка прямого и обратного праймера или 5'-3' / 3'-5'. GC-состав – содержание гуанина (G) и цитозина (C) в ДНК или РНК. GC-состав является одним из ключевых показателей структуры нуклеиновых кислот и играет важную роль в их функционировании. Высокий или низкий GC-состав может влиять на стабильность двойной спирали ДНК, температуру плавления, а также на способность молекул взаимодействовать с другими молекулами, такими как белки или малые молекулы. Молекулярный вес праймера для ПЦР важен по нескольким причинам, например, молекулярный вес праймера влияет на его специфичность к целевой последовательности. Праймеры слишком высокого или низкого молекулярного веса могут неправильно связываться с нежелательными участками ДНК или РНК. Молекулярный вес может также влиять на устойчивость праймера к деградации во время реакции ПЦР. Праймеры с более высоким молекулярным весом могут быть более стабильными и менее подвержены деградации. Температура плавления зависит от состава последовательности ДНК, в частности, от содержания пар G-C и A-T. Обычно температура плавления для кратких праймеров в ПЦР находится в пределах от 50°C до 65°C. Оптимальная температура плавления может быть предварительно оценена с использованием специальных формул, которые учитывают процентное содержание G-C и A-T в последовательности. Однако при планировании реакции ПЦР и выборе температурного режима следует учитывать также другие факторы, такие как специфичность и эффективность гибридизации праймеров с матричной ДНК, а также условия работы используемой ДНК-полимеразы. Температурный профиль ПЦР обычно включает три основные стадии: денатурация, отжиг (аннеалинг) и продление, на каждой из которых используются различные температуры, соответствующие оптимальной температуре плавления праймеров и другим параметрам реакции.

Далее будем использовать базовый язык и библиотеку BioPython для сравнения и лучшего понимания происходящего. BioPython – это библиотека на языке программирования Python, предназначенная для работы с биологическими данных. Она предоставляет широкий спектр функций для анализа и манипулирования данными в области биоинформатики, включая работу с последовательностями ДНК, РНК и белков, а также инструменты для работы с файлами форматов, таких как FASTA, GenBank, BLAST и многих других.

BioPython позволяет легко читать и записывать данные в различных форматах, используемых в биологических исследованиях. Библиотека предоставляет мощные инструменты для работы с последовательностями ДНК, РНК и белков, включая выравнивание, поиск мотивов, трансляцию и многое другое. BioPython включает инструменты для анализа и манипулирования

трехмерными структурами белков и нуклеиновых кислот. Также библиотека предоставляет возможности для работы с различными биологическими базами данных, такими как NCBI и UniProt.

Библиотека может использоваться в сочетании с другими инструментами и библиотеками биоинформатики для выполнения сложных анализов и задач. BioPython – это мощный инструмент для биоинформатических исследований в области геномики, протеомики, структурной биологии и многих других областей биологии. Она широко используется в академических и промышленных исследованиях.

Начнём с простого и будем усложнять, длина последовательности. Код без использования библиотек не требует пояснений, тогда как BioPython использует объект Seq, в который передаётся последовательность ДНК или РНК.

```
def length(dna_str):
    return len(dna_str)

dna_str = "tacgtacgtgtgtcagtcgtgtgtcgat"
print(length(dna_str))
#28

#BioPython
from Bio.Seq import Seq

seq = Seq("tacgtacgtgtgtcagtcgtgtgtcgat")
print(len(seq))
#28
```

Создание комплементарной матрицы ДНК. Образуется путем замены каждого нуклеотида в последовательности ДНК его комплементарным нуклеотидом. При этом также происходит инверсия ориентации, то есть направление чтения структуры изменяется с 3'->5' на 5'->3'. Таким образом, если у вас есть последовательность ДНК, например, 3'-ATCG-5', то комплементарная ей матрица будет иметь вид 5'-TAGC-3'. Обратите внимание, что в BioPython reverse_complement() автоматически создает комплементарную последовательность и инвертирует ее ориентацию.

```
def reverse(dna_str):
    new_dict = {'A':'T', 'T':'A',
            'G':'C', 'C':'G'}

    return ''.join([new_dict[i] for i in dna_str])[::-1]

dna_str = "ATCG"
```

```
print(reverse(dna_str))
#CGAT

#BioPython
from Bio.Seq import Seq

seq = Seq("ATCG")
complement_seq = seq.reverse_complement()
print(complement_seq)
#CGAT
```

Для определения содержания гуанина (G) и цитозина (C) в ДНК или РНК можно использовать обычный строковый счётчик. BioPython же имеет функцию GC(), которая выполняет автоматический подсчёт.

```
def concentration(dna_str):

    g_conc = dna_str.upper().count("G")
    c_conc = dna_str.upper().count("C")

    return round((g_conc+c_conc) * 100 / len(dna_str), 0)

print(concentration('ATCG'))
#50.0

#BioPython

from Bio.SeqUtils import GC # GC- состав
dna_nuc = Seq("ATGC")
print(GC(dna_nuc))
#50.0
```

В BioPython вы можете использовать модуль Bio.SeqUtils для вычисления молекулярного веса белков или нуклеотидных последовательностей. Вот пример кода для вычисления молекулярного веса белка:

```
#BioPython
from Bio.SeqUtils import molecular_weight
print("%0.2f" % molecular_weight("ATCG"))
#1253.80
```

Для выполнения транскрипции нуклеотидной последовательности с использованием BioPython вы можете использовать метод transcribe() объекта Seq. Этот код транскрибирует последовательность ДНК "ATGC" в РНК. Вот пример кода:

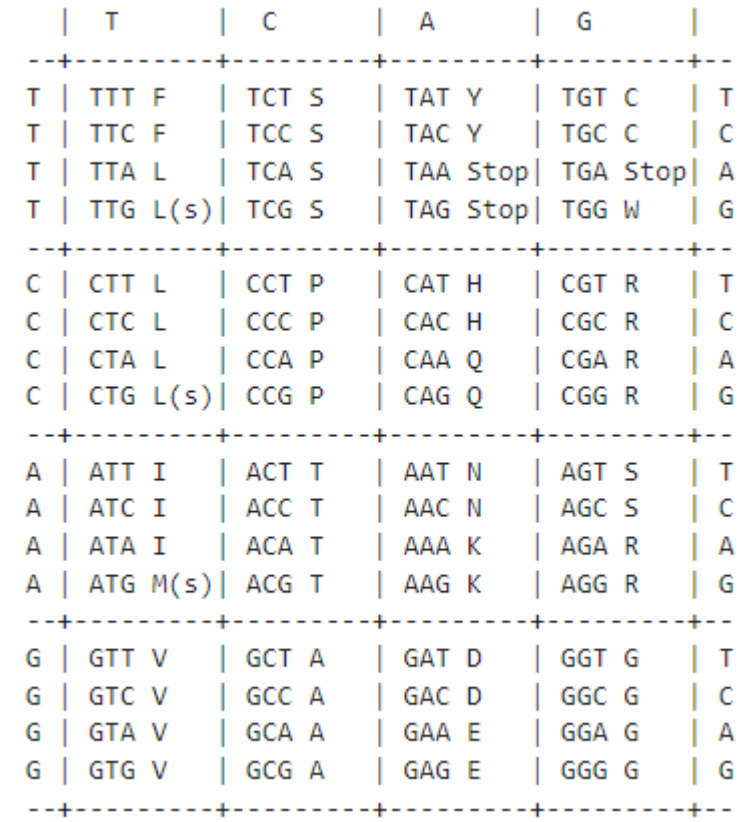
```
def transcriptor(dna_str):

    pattern_dict = {
        "A": "U",
        "G":"C",
        "T":"A",
        "C":"G"
    }

    rna_str = ""

    for nuc in dna_str:
        rna_str += pattern_dict[nuc.upper()]

    return rna_str

dna_str = "ATGC"
print(transcriptor(dna_str))
#'UACG'

#BioPython
from Bio.Seq import transcribe # транскрипция
dna_nuc = Seq("ATGC")
print(transcribe(dna_nuc))
#'UACG'
```

Трансляция процесс синтеза белка на основе молекул РНК. Этот процесс можно разделить на три основные этапа: Инициация – процесс начинается с инициации, когда молекула РНК (мРНК), содержащая информацию о последовательности аминокислот, связывается с рибосомой; элонгация – рибосома перемещается вдоль мРНК, транслируя ее, и добавляет новые аминокислоты к образующейся полипептидной цепи в соответствии с последовательностью кодонов на мРНК; терминация – процесс продолжается до тех пор, пока рибосома не достигнет стоп-кодона на мРНК, который сигнализирует ей о завершении синтеза полипептидной цепи. На этом этапе синтез белка завершается, и полипептидная цепь освобождается.

Последовательность из трёх нуклеотидов – триплет. Каждому триплету соответствует конкретная аминокислота в белке. Триплеты, также известные как кодоны, являются базовыми единицами генетического кода и участвуют в процессе считывания информации с мРНК рибосомой во время трансляции. Получить таблицу соответствий можно с помощью следующего кода:

```
from Bio.Data import CodonTable
table = CodonTable.unambiguous_dna_by_name["Standard"]
```

print(table)



	T	C	A	G	
T	TTT F	TCT S	TAT Y	TGT C	T
T	TTC F	TCC S	TAC Y	TGC C	C
T	TTA L	TCA S	TAA Stop	TGA Stop	A
T	TTG L(s)	TCG S	TAG Stop	TGG W	G
C	CTT L	CCT P	CAT H	CGT R	T
C	CTC L	CCC P	CAC H	CGC R	C
C	CTA L	CCA P	CAA Q	CGA R	A
C	CTG L(s)	CCG P	CAG Q	CGG R	G
A	ATT I	ACT T	AAT N	AGT S	T
A	ATC I	ACC T	AAC N	AGC S	C
A	ATA I	ACA T	AAA K	AGA R	A
A	ATG M(s)	ACG T	AAG K	AGG R	G
G	GTT V	GCT A	GAT D	GGT G	T
G	GTC V	GCC A	GAC D	GGC G	C
G	GTA V	GCA A	GAA E	GGA G	A
G	GTG V	GCG A	GAG E	GGG G	G

Рисунок 3 – Результат выполнения вышенаписанного кода

В генетическом коде существует 64 различных триплета (кодона), каждый из которых кодирует либо конкретную аминокислоту, либо сигналы начала или завершения трансляции. Например, триплет AUG кодирует аминокислоту метионин и также является стартовым кодоном, указывающим начало синтеза белка. Кроме того, есть триплеты, которые являются стоп-кодонами (UAA, UAG, UGA), сигнализирующими о завершении синтеза белка.

1.3. Глобальное и локальное выравнивание

Вопросы:
1. Метод скользящего окна.
2. Глобальное выравнивание.
3. Локальное выравнивание.

Метод скользящего окна – это техника, используемая в анализе последовательностей данных, которая позволяет обнаруживать определенные шаблоны или характеристики в этих данных. Этот метод осуществляет сканирование данных с определенным окном фиксированного размера, которое перемещается по всей последовательности с определенным шагом. В каждом положении окно вычисляет некоторую характеристику или статистику для фрагмента данных, находящегося внутри него.

Например, в биоинформатике метод скользящего окна может использоваться для поиска участков с высокой плотностью определенных нуклеотидных последовательностей или для обнаружения повторяющихся узоров в геноме.

Этот метод может быть полезен, когда вы хотите исследовать данные на наличие локальных структур или шаблонов, которые могут быть упущены при простом обзоре всей последовательности. Он также может быть применен в различных областях, таких как анализ временных рядов, обработка изображений и др.

В биоинформатике метод скользящего окна может помочь обнаружить различные структурные или функциональные участки в биологических последовательностях, что может быть полезно при анализе геномных данных, поиске мотивов связывания белков и многом другом.

Выравнивание последовательностей – процесс поиска схожих областей в двух или более последовательностях (ДНК, РНК или белка). Это позволяет устанавливать сохранённые признаки между видами, учитывать их генетическое родство, отслеживать эволюцию, нахождение консервативных участков, обнаруживать мутации для конкретного вида, предсказывать функциональные элементы.

Попарное выравнивание – методика, которая позволяет сравнивать и выравнивать две последовательностей с целью выявления их сходства и различия. Цель состоит в том, чтобы установить оптимальное расположение последовательностей с целью максимизации совпадений при минимизации несоответствий и вставок

Множественное выравнивание последовательностей – методика выравнивания трёх и более биологических последовательностей. Предполагается, что входной набор последовательностей имеет эволюционную связь. Среди алгоритмов множественного выравнивания последовательностей можно выделить прогрессивные методы (ClustalW и T-Coffee) и итеративные методы (MUSCLE и MAFFT).

Глобальное и локальное выравнивания – это два основных подхода к сравнению последовательностей в биоинформатике. **Глобальное выравнивание** – стратегия выравнивания последовательностей, при которой последовательности выравниваются по всей длине, при этом целью является нахождение наилучшего соответствия, учитывая всю длину последовательностей, от начала до конца

Используется если ожидается, что сравниваемые последовательности имеют большое сходство по всей длине. Одним из наиболее широко используемых алгоритмов для глобального выравнивания является алгоритм Нидлмана-Вунша.

В основе его работы лежит динамическое программирование. Он применяется для сопоставления двух последовательностей путем нахождения наилучшего соответствия между ними с учетом весов или штрафов за различия.

Процесс выравнивания начинается с создания матрицы, в которой строки и столбцы представляют символы из каждой последовательности. Затем заполняются значения в этой матрице, отражающие степень сходства или различия между символами. После этого определяется оптимальный путь через эту матрицу, который обеспечивает наилучшее выравнивание.

Одна из ключевых особенностей алгоритма Нидлмана-Вунша – это то, что он рассматривает все возможные выравнивания и выбирает то, которое обеспечивает максимальное сходство между последовательностями. Он эффективен для выравнивания коротких или средних по длине последовательностей, но может быть неэффективен при работе с длинными последовательностями из-за высокой вычислительной сложности.

В алгоритме Нидлмана-Вунша используются различные штрафы или веса для вставок, удалений и несоответствий символов. Эти параметры позволяют настраивать выравнивание в соответствии с конкретными требованиями задачи анализа. Пример работы алгоритма представлен на рисунке 4. Подставим следующие параметры: вес совпадения (match) = +1; штраф несовпадения/замены (mismath) = -1; штраф за делецию/инсерцию (gap) = -1. Начало алгоритма в левом верхнем углу (с нуля), при этом каждая последовательность содержит значения от –1 до длины последовательности. Рассмотрим первый шаг, имеется 0, -1 (для горизонтального А), -1 (для вертикального А). Из этих значений выбирается максимальное и к нему будет добавляться значение штрафа или поощрения, в данном случае максимальное значение 0, А и А одинаковые нуклеотиды, соответственно в пустую ячейку добавляем значение совпадения + максимальное значение (0+1). Далее движение алгоритма вправо, если совпадение, то к снова к максимальному значению из новой тройки добавляется значение match, в противном случае добавление штрафа. Так проходит алгоритм по всей таблице. После завершения распределения весов алгоритм начинает свою работу в правом нижнем углу, выбираю максимальное значение и в своём окружении.

А

0	A_{-1}	T_{-2}	C_{-3}	C_{-4}	G_{-5}	A_{-6}
A_{-1}	1	0	-1	-2	-3	-2
T_{-2}	0	2	1	0	-1	-2
G_{-3}	-1	1	1	0	1	0
G_{-4}	-2	0	0	0	1	0
T_{-5}	-3	-1	-1	-1	0	0
A_{-6}	-2	-2	-2	-2	-1	1

Б

0	A	T	C	C	G	A
A	↖					
T		↖	←	←		
G					↖	
G					↑	
T					↑	
A						↖

А – Две последовательности по вертикали и горизонтали таблицы, внутри самой таблице распределение значений в зависимости от параметров, Б – Выбор наилучшего пути для сопоставления нуклеотидных последовательностей.

Рисунок 4 – Пример работы алгоритма Нидлмана-Вунша

Глобальное выравнивание в биоинформатике с помощью BioPython может быть реализовано с использованием алгоритма Нидлмана-Вунша. Этот алгоритм доступен в BioPython в модуле Bio.pairwise2 или PairwiseAligner.

```
# попарное сравнение только двух последовательностей
from Bio.Align import PairwiseAligner
# создание объекта последовательности
from Bio.Seq import Seq
# удобная визуализация
from Bio.pairwise2 import format_alignment

# создаём объект класса
aligner = PairwiseAligner()
# передаём в качестве аргументов метода две послдеовательности и вызываем метод
res_aligns = aligner.align("ATCCGA", "ATGGTA")

# иттерируемся по результамам метода
for res_align in res_aligns:
   print(res_align)
```

```
target            0 ATCCG--A 6
                  0 ||--|--| 8
query             0 AT--GGTA 6

target            0 ATCC-G-A 6
                  0 ||---|-| 8
query             0 AT--GGTA 6

target            0 ATC-CG-A 6
                  0 ||---|-| 8
query             0 AT-G-GTA 6

target            0 AT-CCG-A 6
                  0 ||---|-| 8
query             0 ATG--GTA 6

target            0 ATCCG-A 6
                  0 ||.-|-| 7
query             0 ATG-GTA 6

target            0 ATCCG-A 6
                  0 ||-.|-| 7
query             0 AT-GGTA 6
```

Рисунок 5 – Наилучшие совпадения глобального выравнивания

Аналогичную задачу решает модуль Bio.pairwise2 для попарного сравнения только двух последовательностей. Рассмотрим принцип его работы:

```
# попарное сравнение только двух последовательностей
from Bio import pairwise2

# создание экземпляра класса
seq1 = Seq("ATCCGA")
seq2 = Seq("ATGGTA")
res_align = pairwise2.align.globalms(seq1, seq2, 1, -1, -1, -1)

# удобный визуальный вид
print(format_alignment(*res_align[0]))
print(format_alignment(*res_align[1]))
```

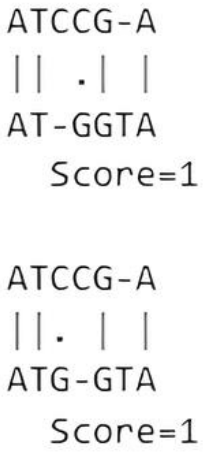


Рисунок 6 – Наилучшие совпадения глобального выравнивания

Алгоритм globalms реализует глобальное выравнивание с использованием матрицы оценок штрафов за совпадения, несовпадения, открытые и расширенные зазоры. Позволяет пользователю определить собственные штрафы за совпадение, несовпадение, вставку и удаление. Этот алгоритм использует динамическое программирование для нахождения оптимального выравнивания.

Алгоритм globalxx также выполняет глобальное выравнивание, но с использованием более простой матрицы оценок, где все совпадения получают одинаковый балл, а все несовпадения и зазоры получают единичный штраф. globalxx может быть полезен для быстрого и простого глобального выравнивания, но менее точен по сравнению с globalms.

Локальное выравнивание – стратегия выравнивания последовательностей, при которой последовательности выравниваются по определённому участку. Цель нахождение наилучшего соответствия,

игнорирую всё остальное. Используется если ожидается, что сравниваемые последовательности имеют сходство только в определённых местах. Алгоритм Смита–Ватермана. Параметры: вес совпадения (match) = +1; вес замены (mismath) = -1; штраф за делецию (gap) = -1. Алгоритм очень похож на Нидлмана-Вунша, различия лишь в том, что значения в ячейке не могу опуститься ниже 0. Пример на рисунке 7.

А

0	T_0	C_0	G_0	G_0	T_0	C_0
A_0	0	0	0	0	0	0
T_0	1	0	0	0	1	0
G_0	0	0	1	1	0	0
G_0	0	0	1	2	1	0
T_0	1	0	0	1	3	2
A_0	0	0	0	0	2	2

Б

0	T	C	G	G	T	C
A						
T						
G			1			
G				2		
T					3	
A						

А – Две последовательности по вертикали и горизонтали таблицы, внутри самой таблице распределение значений в зависимости от параметров, Б – Наилучшее совпадение от максимального значения в таблице к минимальному

Рисунок 7 – Пример работы алгоритма Смита–Ватермана

Пример работы алгоритма Смита-Ватермана также представлен в BioPython. В Biopython есть функция pairwise2.align.localms (localxx) для выполнения локального выравнивания последовательностей. Ниже приведен пример использования этой функции:

```
seq1 = Seq("TCGGTC")
seq2 = Seq("ATGGTA")

res_aligns = pairwise2.align.localms(seq1, seq2, 1, -1, -1, -1)
for res_align in res_aligns:
   print(format_alignment(*res_align))
```

```
3 GGT
  |||
3 GGT
  Score=3
```

Рисунок 8 – Наилучшее совпадения локального выравнивания

Ещё один пример рассмотрим для поиска конкретного участка в большей последовательности:

```
seq1="cagtgctagctagcatgcctgctcctcaaccttccaggctcgagacatcatgtcatgctagctgtcagtca"
seq2 = "cctcaaccttccag"

res_aligns = pairwise2.align.localms(seq1, seq2, 3, -2, -2, -2)
for res_align in res_aligns:
   print(format_alignment(*res_align))
```

```
24 CCTCAACCTTCCAG
   ||||||||||||||
 1 CCTCAACCTTCCAG
  Score=42
```

Рисунок 9 – Наилучшее совпадения локального выравнивания

Глобальные выравнивания полезны для сравнения двух последовательностей в целом и выявления их общих структур и функций. Локальные выравнивания, с другой стороны, подходят для поиска конкретных областей с высокой степенью сходства, что может быть полезно при обнаружении гомологичных регионов в белках или геномах, а также при анализе делеций, вставок и мутаций.

Марковские цепи играют важную роль в молекулярной генетике, особенно в анализе последовательностей ДНК и белков. В контексте биоинформатики они могут быть использованы для моделирования эволюционных процессов, предсказания структуры белков, анализа генных последовательностей и других биологических данных.

Марковская модель представляет собой математическую модель, описывающую последовательность событий, где вероятность каждого события зависит только от предыдущего состояния. В случае анализа последовательностей ДНК или белков, состояния могут представлять собой конкретные нуклеотиды или аминокислоты, а вероятности перехода между состояниями могут быть определены на основе известных данных об эволюции или экспериментальных наблюдений.

BioPython предоставляет инструменты для работы с марковскими моделями, что делает их доступными для различных задач анализа биологических данных. Это включает в себя возможность создания, обучения и

использования марковских моделей для выравнивания последовательностей, предсказания структуры белков, анализа генных последовательностей и других биологических данных.

1.4. Алгоритмы работы с последовательностями

Вопросы:
1. Алгоритм Бойера-Мура.
2. Преобразование Барроуза-Уилера.
3. Хеш-таблицы.
4. Поиск открытых рамок считывания.

Алгоритм Бойера-Мура – это эффективный алгоритм для поиска подстроки в строке. Он используется в биоинформатике для поиска генетических последовательностей в больших геномах. Принцип работы алгоритма Бойера-Мура основан на том, что он проводит поиск справа налево и использует информацию о распределении символов в образце и тексте для оптимизации поиска. Алгоритм сравнивает символы образца и текста справа налево и сдвигает образец наиболее эффективно в случае несовпадения.

В биоинформатике алгоритм Бойера-Мура может использоваться, например, для поиска консервативных участков в последовательностях ДНК или белковых мотивов. Вот примерный код на Python для реализации алгоритма Бойера-Мура:

```
def boyer_moore(text, pattern):
    # Подготовка таблицы смещений
    skip = {}
    for i in range(len(pattern)):
        skip[pattern[i]] = len(pattern) - i - 1

    # Начальные значения
    i = len(pattern) - 1
    while i < len(text):
        j = len(pattern) - 1
        k = i
        while j >= 0 and text[k] == pattern[j]:
            j -= 1
            k -= 1
        if j == -1:
            return k + 1  # Найдено совпадение
        i += skip.get(text[i], len(pattern))
    return -1  # Совпадение не найдено

# Пример использования
```

```
text = "CGTGCCTACTTACTTACAT"
pattern = "TACTTAC"
result = boyer_moore(text, pattern)
if result != -1:
    print(f"Совпадение найдено в позиции {result}.")
else:
    print("Совпадение не найдено.")
#Совпадение найдено в позиции 6.
```

Здесь text представляет собой строку, в которой происходит поиск, а pattern – образец, который мы ищем. Функция boyer_moore возвращает позицию первого совпадения или -1, если совпадений нет.

Преобразование Барроуза-Уилера (BWT) является важным преобразованием в алгоритмах сжатия данных и в биоинформатике для работы с последовательностями ДНК. Это преобразование переставляет символы строки таким образом, что образуется новая строка, а также индекс символа, который предшествует каждой строке в отсортированной версии исходной строки. Это преобразование может быть полезным для поиска подстрок в строке. Алгоритмы, вырожденные из Б-У: bowtie1/2, SOAP2, bwa-sw.

Рассмотрим два варианта сжатия строки «ACCCAAACC». Первый вариант – «A1C3A3C2». Самый простой способ, но эффективность низкая. Второй вариант – «A4C5» – способ лучше, но сложнее, так как необходимо запомнить места элементов в строке. Это и есть преобразование Барроуза-Уилера.

Собственно этапы преобразования Барроуза-Уилера:

1. Отмечается начальный символ;
2. Перебор всех возможных вариантов строки;
3. Лексикографическая сортировка;
4. Запись конечных индексов всех вариантов;
5. Построение FM-индексов;
6. Закодированный/архивированный массив.

Рассмотрим пример кода преобразования Барроуза-Уилера:

```
#исходная строка
name = 'ACTGGACACATTACGAA'
#добавляем в конец строки символ
name = name + '$'
# все возможные варианты
unsort_list = [name[i:]+name[:i] for i in range(len(name))]
print(unsort_list)

#['ACTGGACACATTACGAA$',
 'CTGGACACATTACGAA$A',
 'TGGACACATTACGAA$AC',
```

```
 'GGACACATTACGAA$ACT',
 'GACACATTACGAA$ACTG',
 'ACACATTACGAA$ACTGG',
 'CACATTACGAA$ACTGGA',
 'ACATTACGAA$ACTGGAC',
 'CATTACGAA$ACTGGACA',
 'ATTACGAA$ACTGGACAC',
 'TTACGAA$ACTGGACACA',
 'TACGAA$ACTGGACACAT',
 'ACGAA$ACTGGACACATT',
 'CGAA$ACTGGACACATTA',
 'GAA$ACTGGACACATTAC',
 'AA$ACTGGACACATTACG',
 'A$ACTGGACACATTACGA',
 '$ACTGGACACATTACGAA']

# сортируем список
sort_list = sorted(unsort_list)
print(sort_list)

#['$ACTGGACACATTACGAA',
 'A$ACTGGACACATTACGA',
 'AA$ACTGGACACATTACG',
 'ACACATTACGAA$ACTGG',
 'ACATTACGAA$ACTGGAC',
 'ACGAA$ACTGGACACATT',
 'ACTGGACACATTACGAA$',
 'ATTACGAA$ACTGGACAC',
 'CACATTACGAA$ACTGGA',
 'CATTACGAA$ACTGGACA',
 'CGAA$ACTGGACACATTA',
 'CTGGACACATTACGAA$A',
 'GAA$ACTGGACACATTAC',
 'GACACATTACGAA$ACTG',
 'GGACACATTACGAA$ACT',
 'TACGAA$ACTGGACACAT',
 'TGGACACATTACGAA$AC',
 'TTACGAA$ACTGGACACA']

#последние символы в каждой строке
last_sym = [i[-1:] for i in sort_list]
bw = "".join(last_sym)
print(bw)
```

```
#далее будем работать только с этой строкой, не прибегая к прошлым данным
#'AAGGCT$CAAAACGTTCA'
#или же данную строку можно представить в виде архива
#'A2G2CT$CA4CGT2CA'


#новый список с пустыми значениями
new_list = [''] * len(bw)

for i in range(len(bw)):
# добавление символа в список
   new_list = [bw[i] + new_list[i] for i in range(len(bw))]
   print(new_list)
# сортировка списка
   new_list = sorted(new_list)
   print(new_list)

#пример первых четырёх списков в выводе
#['A', 'A', 'G', 'G', 'C', 'T', '$', 'C', 'A', 'A', 'A', 'A', 'C', 'G', 'T', 'T', 'C', 'A']
['$', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'G', 'G', 'G', 'T', 'T', 'T']
['A$', 'AA', 'GA', 'GA', 'CA', 'TA', '$A', 'CA', 'AC', 'AC', 'AC', 'AC', 'CG', 'GG', 'TG', 'TT', 'CT', 'AT']
['$A', 'A$', 'AA', 'AC', 'AC', 'AC', 'AC', 'AT', 'CA', 'CA', 'CG', 'CT', 'GA', 'GA', 'GG', 'TA', 'TG', 'TT']

#итоговый список будет
print(new_list)

#['$ACTGGACACATTACGAA',
 'A$ACTGGACACATTACGA',
 'AA$ACTGGACACATTACG',
 'ACACATTACGAA$ACTGG',
 'ACATTACGAA$ACTGGAC',
 'ACGAA$ACTGGACACATT',
 'ACTGGACACATTACGAA$',
 'ATTACGAA$ACTGGACAC',
 'CACATTACGAA$ACTGGA',
 'CATTACGAA$ACTGGACA',
 'CGAA$ACTGGACACATTA',
 'CTGGACACATTACGAA$A',
 'GAA$ACTGGACACATTAC',
 'GACACATTACGAA$ACTG',
 'GGACACATTACGAA$ACT',
```

```
 'TACGAA$ACTGGACACAT',
 'TGGACACATTACGAA$AC',
 'TTACGAA$ACTGGACACA']

#в этом списке имеется исходная строка, которая заканчивается на $
# нахождение изначальной последовательности
decoder_bw = [el for el in new_list if el.endswith("$")][0]

print(decoder_bw[:-1])

#ACTGGACACATTACGAA
```

Преобразование Барроуза-Уилера может быть полезным для реализации алгоритмов сжатия данных, таких как алгоритм Барроуза-Уилера (BWT) с алгоритмом Хаффмана или алгоритмом Move-to-Front, а также для поиска подстрок и других задач обработки строк.

Такие упорядоченные структуры гораздо быстрее искать в базе данных, так как они занимают меньше места, однако есть возможность ускорить этот процесс, использую, например, хеш-таблицы. **Хеш-таблицы** – это эффективная структура данных для индексации, особенно при работе с большими объемами данных. Они позволяют быстро находить элементы по ключу, используя хеш-функцию для преобразования ключа в индекс таблицы.

Хеш-функция принимает входные данные (например, строку или число) и возвращает уникальный хеш-код, который затем используется в качестве индекса в таблице. Одинаковые входные данные всегда будут давать одинаковый хеш-код, что обеспечивает быстрый доступ к данным.

Однако стоит помнить, что возможны коллизии, когда два разных ключа дают одинаковый хеш-код. В этом случае используются различные методы разрешения коллизий, такие как метод цепочек или открытая адресация.

Хеш-таблицы широко используются в различных областях, включая базы данных, поиск и индексацию данных, криптографию и многое другое. Их эффективность и скорость доступа делают их незаменимыми при работе с большими объемами данных. Допустим есть список [«data», «name», «para», «are»], для того, чтобы найти элемент «para» необходимо пройтись по списку и сравнить элементы. Если элемент в конце списка, то такой способ будет время затратным, было бы гораздо проще, если был известен индекс элемента. Это как раз можно сделать при помощи хеш-таблиц. Рассмотрим пример: имеется тот же список, однако каждый элемент мы расставим в списке по хеш-функции, которая будет брать каждую букву из элемента, переводить её в кодировку ASCII, сохраняя остаток от деления суммы кодировок на количество элементов. Таким образ каждый элемент в списке будет иметь индекс, привязанный к его данным. В случае поиска элемента в списке, будет аналогично применяться хеш-функцию к элементу, которая вернёт индекс, по которому может находится искомый элемент. Пример ниже на рисунке 10.

data = 100+97+116+97 = 410%4 = 2

name = 110+97+109+101 = 417%4 = 1

para = 112+97+114+97 = 420%4 = 0

are = 97+114+104 = 315%4 = 3

para	name	data	are
0	1	2	3

Рисунок 10 – Пример хеш-таблицы

Однако есть вероятность, что новый элемент списка может уже занятый индекс – коллизия.

Принцип Pigeonhole (принцип "Голубиная дыра") – это простой, но важный принцип комбинаторики, который утверждает, что если больше объектов (голубей) должны быть размещены в меньшем числе контейнеров (гнезд), чем количество объектов, то как минимум один контейнер должен содержать более одного объекта.

Формально принцип Pigeonhole можно сформулировать так: если n объектов размещаются в m контейнерах, где $n>m$, то как минимум один контейнер содержит не менее двух объектов.

Этот принцип широко используется в различных областях, таких как математика, информатика, криптография и другие. Например, его можно применить при анализе коллизий в хеш-таблицах: если количество ключей (объектов) превышает количество доступных ячеек (контейнеров), то по крайней мере одна ячейка будет содержать более одного ключа, что приводит к коллизии. Существует несколько способов борьбы с коллизиями в хеш-таблицах. Вот некоторые из них:

При методе Chaining каждая ячейка хеш-таблицы содержит список элементов, которые имеют одинаковый хеш-код. Когда происходит коллизия, новый элемент добавляется в соответствующий список. Этот метод прост в реализации и эффективен, если коллизии редки. Однако он требует дополнительного места для хранения указателей на списки.

Open Addressing – если происходит коллизия, алгоритм поиска альтернативного слота в таблице для вставки элемента. В отличие от метода цепочек, он не использует дополнительную структуру данных для хранения элементов. Существуют различные стратегии поиска альтернативного слота, такие как линейное зондирование, квадратичное зондирование и двойное хеширование.

Linear Probing – если происходит коллизия, алгоритм проверяет следующую ячейку в таблице. Если она занята, он переходит к следующей и так далее, пока не найдет свободную ячейку. Этот процесс продолжается до тех пор, пока не будет найдено свободное место или не будет достигнут конец таблицы.

Quadratic Probing – алгоритм пробует ячейки с шагом, который увеличивается квадратично (например, 1, 4, 9, 16 и т. д.). Это помогает избежать слишком близких друг к другу кластеров при коллизиях, которые могут возникнуть при линейном зондировании.

Double Hashing – используется вторая хеш-функция для вычисления дополнительного сдвига при коллизии. Это может помочь избежать образования кластеров при определенных последовательностях коллизий.

Выбор конкретного метода зависит от конкретных требований вашего приложения, структуры данных и предполагаемой степени загруженности таблицы.

Поиск открытых рамок считывания (Open Reading Frames, ORFs) и кодирующих фрагментов является важным этапом анализа геномных данных. ORFs представляют собой участки ДНК или РНК, которые могут быть транслированы в белок. Кодирующие фрагменты представляют собой участки генома, содержащие гены или функциональные элементы.

Существует несколько подходов к поиску ORFs и кодирующих фрагментов:

- Алгоритмы поиска ORFs: Эти алгоритмы сканируют геном на наличие потенциальных ORFs, которые начинаются с старт-кодона (например, ATG у эукариот или AUG у прокариот) и заканчиваются стоп-кодоном (TAA, TAG или TGA). После нахождения потенциальных ORFs их длина и содержание аминокислот определяются для оценки вероятности того, что они представляют собой настоящие гены.
- Использование баз данных генов: Многие исследователи используют базы данных генов для сравнения последовательностей с уже известными генами. Это помогает идентифицировать кодирующие фрагменты и присвоить им функции на основе сходства с известными генами или белками.
- Методы машинного обучения: Некоторые исследования применяют методы машинного обучения для обучения моделей на основе известных геномов и последующего прогнозирования расположения генов и ORFs в новых геномных последовательностях.
- Специализированные инструменты и программное обеспечение: Существует множество специализированных инструментов и программ для анализа геномных данных, которые включают в себя функции поиска ORFs и кодирующих фрагментов.

Выбор конкретного метода зависит от характеристик конкретного исследования, доступных ресурсов и специфики геномной последовательности, которую требуется анализировать. Вот пример простого кода на Python с использованием библиотеки BioPython для поиска открытых рамок считывания (ORFs) в геномной последовательности:

```
from Bio.Seq import Seq

def find_orfs(sequence):
    orfs = []
    for frame in range(3):
        start_indices = [i for i in range(len(sequence[frame:])) if sequence[frame
+ i:frame + i + 3] == "ATG"]
        for start_index in start_indices:
            stop_index = frame + start_index
            while stop_index + 3 <= len(sequence):
                codon = sequence[stop_index:stop_index + 3]
                if codon in ["TAA", "TAG", "TGA"]:
                    orfs.append((frame + start_index, stop_index + 3))
                    break
                stop_index += 3
    return orfs

# Пример использования:
dna_sequence = Seq("ATGCGAATGTAGCATCAAA")
orfs = find_orfs(dna_sequence)
print("Найденные ORFs:")
for start, end in orfs:
    print("Старт: {}, Стоп: {}, Длина: {}".format(start, end, end - start))

# Найденные ORFs:
Старт: 0, Стоп: 12, Длина: 12
Старт: 6, Стоп: 12, Длина: 6
Старт: 6, Стоп: 12, Длина: 6
Старт: 6, Стоп: 12, Длина: 6
```

Этот код сканирует геномную последовательность на наличие потенциальных ORFs, начиная с каждой из трех рамок считывания (фаз). Затем он находит стартовые кодоны ("ATG") и ищет следующий стоп-кодон ("TAA", "TAG", "TGA") в той же фазе считывания. Если такой кодон найден, ORF добавляется в список.

1.5. Протеомный анализ

Вопросы:
1. Цели и задачи протеомики.
2. Количественный и качественный анализ протеома.
3. Масс-спектрометрия.

В этом разделе мы рассмотрим теоретические основы протеомного анализа. Белки – это биомолекулы, состоящие из длинных цепей аминокислот, связанных между собой пептидными связями. Структура белков может быть описана на нескольких уровнях организации. Первичная структура – это последовательность аминокислот в полипептидной цепи белка. Первичная структура определяется генетической информацией и является основой для дальнейшей структуры белка. Вторичная структура – это пространственное расположение аминокислот в полипептидной цепи белка. Вторичная структура может быть α-спиралью, β-складкой или другими типами вторичной структуры, образующимися за счет водородных связей между аминокислотами. Третичная структура – это трехмерное пространственное строение белка, которое определяется взаимодействиями между различными участками полипептидной цепи, такими как водородные связи, гидрофобные взаимодействия, сольватация и дисульфидные мостики. Кватерническая структура – это организация двух или более полипептидных цепей (субъединиц) в сложную трехмерную структуру белка. Кватерническая структура характерна для многих белков, состоящих из нескольких субъединиц.

Структура белков определяет их функцию, поэтому понимание структуры белков помогает в понимании их биологических функций и в разработке методов для их модификации и использования в медицинских и биотехнологических целях.

Протеомика – это область науки, изучающая структуру и функции всех белков, которые присутствуют в клетке или организме. Целями и задачами протеомики являются:

1. Исследование белковых взаимодействий. Протеомика позволяет изучать взаимодействия между различными белками в клетке, что помогает понять их роль в клеточных процессах.

2. Идентификация и характеризация белков. Протеомика позволяет идентифицировать и описать все белки, которые присутствуют в клетке или организме, что помогает лучше понять их функции.

3. Изучение изменений в протеоме. Протеомика позволяет выявлять изменения в составе и уровне экспрессии белков при различных физиологических или патологических условиях, что может помочь в диагностике и лечении различных заболеваний.

4. Развитие новых методов анализа белков. Протеомика способствует развитию новых технологий и методов анализа белков, что делает возможным более точное и полное изучение протеома.

5. Поиск новых потенциальных мишеней для лекарственных препаратов. Изучение протеома может помочь выявить новые цели для лекарственных препаратов и разработать более эффективные методы лечения различных заболеваний.

Качественный анализ протеома направлен на идентификацию всех белков, присутствующих в определенной клетке, ткани или организме. Для этого используются высокопроизводительные методы, такие как жидкостная

хроматография-масс-спектрометрия (LC-MS), масс-спектрометрия (MS), Western blotting или иммунофлуоресцентная микроскопия. Качественный анализ протеома помогает понять биологические процессы, происходящие в клетках, и идентифицировать потенциальные маркеры болезни или терапевтические цели.

Количественный анализ протеома позволяет определить количество конкретных белков или их изменения в различных условиях или состояниях, например, при развитии болезни или в ответ на лечение. Этот анализ может быть проведен с использованием методов, таких как LC-MS, изотопная маркировка или гель-электрофорез.

Масс-спектрометрия – это метод анализа веществ по массе и ионному заряду их атомов и молекул. В процессе масс-спектрометрии пробы исследуемого вещества расщепляются на ионы под действием ионизирующего излучения, после чего эти ионы разделяются по массе в масс-анализаторе. Полученные данные об ионах и их массах позволяют определить химический состав и структуру анализируемого образца. Масс-спектрометрия является мощным методом анализа, который широко используется в химии, биохимии, физике, медицине и других областях науки.

В общем, принцип работы масс-спектрометра включает следующие этапы: заряженные ионы превращаются в нейтральные атомы и молекулы; после ионизации заряженные частицы переносятся в масс-анализатор, где происходит сортировка ионов по соотношению массы и заряда; в качестве детекторов используются динодные вторично-электронные умножители либо фотоумножители.

Анализ масс-спектров пептидов с помощью программного обеспечения является важным этапом в исследовании белков и их функций. Для этого используются различные программы, которые помогают идентифицировать пептиды и определять их массу.

Программное обеспечение для анализа масс-спектров пептидов может быть бесплатным или платным и иметь различные функции. Некоторые из них позволяют проводить поиск в базах данных белков, анализировать фрагменты пептидов, определять их массу и состав аминокислот, а также проводить квантификацию белков.

Некоторые из наиболее популярных программ для анализа масс-спектров пептидов включают Mascot, Proteome Discoverer, MaxQuant, PEAKS и другие. Эти программы имеют широкие возможности для анализа и интерпретации масс-спектров пептидов и являются незаменимыми инструментами в протеомике и масс-спектрометрии.

В целом, анализ масс-спектров пептидов с помощью программного обеспечения позволяет исследователям получать более точные и надежные данные о структуре белков и их функциях, что является важным в научном исследовании и медицине.

1.6. Пространственная структура белковых молекул

Вопросы:
1. Методы получения пространственной структуры белковых молекул.
2. Примеры искусственного интелекта для предсказания структуры белка.

Рассмотрим ключевые методы к получению пространственной структуре белковых молекул. Один из них – рентгеноструктурный анализ. Это метод анализа кристаллических материалов, основанный на дифракции рентгеновских лучей на атомах материала. С его помощью можно определить строение кристаллической решетки, расположение атомов в кристалле, а также получить информацию о межатомных расстояниях и углах в кристалле. Рентгеноструктурный анализ широко применяется в различных областях науки и техники, таких как химия, физика, материаловедение, биология и др.

Ядерно-магнитный резонанс (ЯМР) – это метод, который позволяет изучать структуру и динамику белковых молекул. ЯМР анализирует взаимодействие ядер атомов в молекуле под действием магнитного поля, что позволяет получить информацию о их расположении и движении.

Для белковых молекул ЯМР представляет огромное значение, так как позволяет исследовать их структуру в растворе без необходимости кристаллизации. Это особенно важно, поскольку многие белки не могут быть кристаллизованы, что может затруднять изучение их структуры другими методами.

С помощью ЯМР исследователи могут определить 3D структуру белковых молекул, исследовать их внутреннюю динамику и взаимодействия с другими молекулами. Это позволяет лучше понять функции белков и их роль в клеточных процессах, что имеет огромное значение для развития новых препаратов и лечебных методов.

Крио-электронная микроскопия – это метод изучения биологических образцов, таких как белки, при очень низких температурах. Этот метод позволяет сохранить структуру образца близкой к естественной, что позволяет увидеть детали на молекулярном уровне. В случае белков, крио-электронная микроскопия может использоваться для определения их трехмерной структуры. Это важно для понимания их функций и механизмов действия в организме.

Преимущества крио-электронной микроскопии включают в себя возможность изучения образцов без фиксации и окрашивания, что исключает исказения структуры, и возможность получения высокого разрешения изображений. Этот метод позволяет получить трехмерные структуры белковых молекул с разрешением в несколько ангстремов.

Компьютерное моделирование для пространственной структуры белков – это процесс, при котором используются различные методы и алгоритмы для предсказания трехмерной структуры белков на основе их аминокислотной последовательности. Такие методы включают в себя молекулярное

моделирование, молекулярную динамику, а также методы искусственного интеллекта, такие как машинное обучение и нейронные сети.

Остановимся чуть подробнее на последнем. Одним из итоговых вариантов получения пространственной структуры белка является файл, который содержит координаты расположения каждого атома в молекуле. Одним из таких форматов файла является PDB. Формат PDB (Protein Data Bank) является стандартным форматом для хранения информации о белковых структурах, координатах атомов и их взаимодействиях. PDB файлы содержат данные о трехмерной структуре белковых молекул, включая информацию о расположении атомов, связей и других важных характеристиках структуры.

Файлы в формате PDB можно открыть и просмотреть специализированными программами для работы с белковыми структурами, такими как PyMOL, Chimera, VMD и другими. Такие программы позволяют анализировать и визуализировать белковые структуры, исследовать их взаимодействия с другими молекулами, моделировать изменения в структуре и многое другое. Пример работы с подобными форматами описаны кодом ниже.

```
from Bio.PDB import *
from Bio import SeqIO
from Bio.SeqUtils.ProtParam import ProteinAnalysis

#загрузка файла из базы данных
pdb = PDBList()
pdb.retrieve_pdb_file('4LB9',pdir='4LB9')

#получение пептидной последовательности
cif = MMCIFParser()
structure = cif.get_structure('ALBM', '4LB9.cif')
polypeptides = CaPPBuilder()
pol_obj = polypeptides.build_peptides(structure)
seq = pol_obj[0].get_sequence()
print(f'Length: {len(seq)} ', '\n', seq)

#Length: 583
AHKSEVAHRFKDLGEENFKALVLIAFAQYLQQCPFEDHVKLVNEVTEFAKT
CVADESAENCDKSLHTLFGDKLCTVATLRETYGEMADCCAKQEPERNECFL
QHKDDNPNLPRLVRPEVDVMCTAFHDNEETFLKKYLYEIARRHPYFYAPELL
FFAKRYKAAFTECCQAADKAACLLPKLDELRDEGKASSAKQRLKCASLQKF
GERAFKAWAVARLSQRFPKAEFAEVSKLVTDLTKVHTECCHGDLLECADDR
ADLAKYICENQDSISSKLKECCEKPLLEKSHCIAEVENDEMPADLPSLAADFV
ESKDVCKNYAEAKDVFLGMFLYEYARRHPDYSVVLLLRLAKTYETTLEKCCA
AADPHECYAKVFDEFKPLVEEPQNLIKQNCELFEQLGEYKFQNALLVRYTK
KVPQVSTPTLVEVSRNLGKVGSKCCKHPEAKRMPCAEDYLSVVLNQLCVLH
EKTPVSDRVTKCCTESLVNRRPCFSALEVDETYVPKEFNAETFTFHADICTLS
```

```
EKERQIKKQTALVELVKHKPKATKEQLKAVMDDFAAFVEKCCKADDKETC
FAEEGKKLVAASQAALG

seq_analysis = ProteinAnalysis(seq)
print(seq_analysis.molecular_weight())
#66243.27700000019
print(seq_analysis.count_amino_acids())
#{'A': 62,
 'C': 35,
 'D': 35,
 'E': 62,
 'F': 31,
 'G': 12,
 'H': 16,
 'I': 8,
 'K': 59,
 'L': 60,
 'M': 6,
 'N': 17,
 'P': 24,
 'Q': 20,
 'R': 24,
 'S': 24,
 'T': 28,
 'V': 41,
 'W': 1,
 'Y': 18}
print(seq_analysis.isoelectric_point())
#5.731098365783692
print(seq_analysis.charge_at_pH(7.5))
#-15.227970766350154
print(seq_analysis.charge_at_pH(5.7))
#0.5333236908958128
print(seq_analysis.aromaticity())
#0.08576329331046312
```

Этот самый формат можно получить с помощью вышеописанных методов, но нас интересует системы на основе искусственных нейронных сетей. Вот несколько примеров использования искусственного интеллекта для предсказания пространственной структуры белка:

AlphaFold: Это инструмент, основанный на искусственном интеллекте, который широко признан за свою высокую точность и универсальность в предсказании 3D структуры белков. AlphaFold полезен для понимания отношений между структурой и функцией на основе моделей 3D структуры

белка и может служить шаблоном или референсом для экспериментального структурного анализа, включая рентгеновскую кристаллографию, ЯМР и крио-ЭМ анализ. Исследователи в настоящее время изучают весь потенциал моделей белка, созданных с помощью AlphaFold. Предсказание степени тяжести заболевания, вызванного миссенс-мутациями, является одним из таких применений.

DeepMind’s AlphaFold Database: Исследователи использовали AlphaFold для предсказания структур более чем 200 миллионов белков из примерно 1 миллиона видов, охватывая почти все известные белки на планете. Эта база данных доступна бесплатно и была создана DeepMind (компанией, разработавшей AlphaFold) и Европейским институтом биоинформатики Европейской лаборатории молекулярной биологии (EMBL–EBI).

AI-Driven Approaches: Подходы, основанные на искусственном интеллекте, такие как моделирование по гомологии и методы ab initio, значительно улучшили предсказание структуры белка. Алгоритмы машинного обучения могут изучать большие базы данных экспериментально определенных структур белка, чтобы определить закономерности и связи между последовательностями белка и их соответствующими структурами.

Данным системам достаточно знать аминокислотную последовательность, которая будет использована для синтеза белковой молекулы. Далее система на основе обучения на огромных базах данных предлагает наиболее вероятный вариант скручивания пространственной структуры белка.

1.7. Анализ биомедицинских изображений. Феномика

Вопросы:
1. Цели и задачи анализа изображений.
2. Методы анализа имиджей.
3. Предобработка имиджей.

Анализ изображений в биологии представляет собой метод исследования, включающий использование компьютерной обработки данных и алгоритмов для анализа изображений, полученных из биологических образцов. Этот метод позволяет ученым изучать и анализировать сложные структуры и процессы, которые не всегда могут быть видны невооруженным глазом.

Он может быть использован для изучения клеток, тканей, органов, организмов, а также для анализа динамики процессов, таких как деление клеток, миграция клеток, взаимодействие белков и многое другое. С помощью анализа изображений можно получить количественные данные, которые помогают ученым лучше понять биологические процессы и их изменения при различных условиях.

Примеры методов анализа изображений в биологии включают сегментацию изображений (разделение изображения на отдельные объекты), измерение размеров и формы клеток, отслеживание движения клеток,

определение светимости и интенсивности сигналов и многое другое. Анализ изображений в биологии помогает ученым генерировать новые гипотезы, проверять их с помощью экспериментов и расширять наше знание о живых организмах. К задачам анализа изображений относятся: идентификация и классификация организмов – анализ изображений позволяет определить вид или подвид организма, что может быть полезно для научных исследований, охраны природы, сельского хозяйства и медицины; мониторинг и оценка популяций – анализ изображений позволяет отслеживать изменения в популяциях организмов, такие как количество особей, распределение или состояние здоровья, что важно для оценки экосистем и выявления угроз для биоразнообразия; изучение поведения организмов – анализ изображений позволяет наблюдать и анализировать поведенческие особенности организмов, такие как движение, коммуникация, питание и размножение, что может помочь понять их процессы адаптации и взаимодействия в естественной среде; диагностика и исследования заболеваний: анализ изображений позволяет диагностировать и изучать заболевания организмов, например, путем анализа морфологических изменений в тканях и клетках, что помогает в медицинских исследованиях и разработке лечения; генетические исследования – анализ изображений может использоваться для изучения генетических характеристик организмов, например, для идентификации генетических мутаций или определения полиморфизмов, что важно для генетических исследований и селекции.

В последние два десятилетия сформировалась новая омиксная область биологии – феномика. В приложении к биологии растений (феномика растений) фокусируется на выявлении закономерностей организации и изменениях физических и биохимических характеристик растений, рассматриваемых как совокупность фенотипов растительного организма. Феномика является дисциплиной, которая активно использует достижения геномной эры и биоинформатики, чтобы предоставить стандартизированный и статистически значимый фактический материал о фенотипах с высокой степенью детализации. Получение и анализ информации о фенотипах в феномике получило название фенотипирование.

В настоящее время широко распространено высокопроизводительное фенотипирование, которое обеспечивает цифровой автоматизированный анализ больших выборок данных. Полученные с использованием феномных технологий массивы данных регистрируются и обрабатываются автоматически, что лишает их проблем субъективной оценки и неадекватной статистической обработки.

Цифровое фенотипирование – это процесс использования современных цифровых технологий для анализа и измерения фенотипических характеристик растений на основе их изображений. Цифровое фенотипирование позволяет проводить более точный и объективный анализ, а также быстрее обрабатывать большие объемы данных образцов растений. Такие данные могут быть использованы для селекции растений, изучения их реакции на различные

условия выращивания и для разработки устойчивых культур с улучшенными качествами и урожайностью.

Одним из важнейших свойств фенотипа является его многоуровневый характер, поскольку проявление генома можно описать на всех уровнях организации живых систем – от молекулярного, до целого организма. Фенотипирование органов растений можно разделить на фенотипирование подземных и надземных органов. Надземные растительные фенотипы подразделяются на три категории: структурные, физиологические и временные. К структурным фенотипам относится морфология растений, к физиологическим – рост и обмен веществ. Временные фенотипы основаны на траекториях роста растений и временных событиях в их жизни, таких как прорастание, появление репродуктивных органов и старение. Каждая из категорий дополнительно подразделяется на холистические и компонентные фенотипы. Холистические фенотипы рассматривают всё растение целиком для оценки его формы, в то время как компонентные фенотипы изучают отдельные части растения, такие как листья, стебли, цветы и плоды.

Фенотипирование листьев. Листья выполняют множество важных функций для растений, такие как фотосинтез, дыхание, транспирация и гуттация, а также помогают растениям взаимодействовать с окружающей средой. У цветковых растений листья демонстрируют большое разнообразие. При этом внешние характеристики листа такие как размер, форма, опушение являются одними из базовых для видовой идентификации растения. Анализ формы и размеров листа по изображению можно оценивать полуавтоматически, с использованием таких пакетов, как ImageJ.

На основе количественных изменений площади листьев возможно построение моделей формирования и развития розетки арабидопсиса. При этом появляется возможность выявления центральных генетических механизмов, участвующие в этом процессе, а также взаимосвязей между увеличением площади листа, фотосинтезом и накоплением сырой массы. Для определения расположения и ориентации листа на стебле, а также изучения циркадных ритмов растений, применяются технологии лазерного сканирования и цифровые стереокамеры для создания 3D-моделей растений.

Устьица и трихомы – микроскопическая структуры поверхности – также является объектами фенотипирования. При анализе необходимо оценивать количество, геометрические характеристики и распределение устьиц и трихом на поверхности листовой пластинки. Визуальный подсчет трихом с использованием светового микроскопа очень трудоемок и, кроме количества трихом, не позволяет анализировать их форме и размер.

Фенотипирование побегов. Оценка сырой массы растения является одним из ключевых показателей его роста и развития. Однако для определения живой или сухой массы растения необходимо удалить его из почвы. Это ограничивает возможности оценки массы растения в процессе его роста. При использовании методов высокопроизводительного фенотипирования можно успешно определять сырую массу растения на основе неинвазивногцо анализа

изображений побега. Например, для оценки сырой массы арабидопсиса используется анализ площади розетки на изображении, полученном сверху. Кроме того, методы анализа изображений позволяют оценивать не только сырую массу, но и различные характеристики формы розетки, динамику роста и другие параметры. В некоторых работах информация о видимом изображении розетки комбинируется с другими фенотипическими параметрами, такими как содержание хлорофилла, оцениваемое по флуоресценции. Для злаковых растений проводится несколько цифровых снимков для установления сырой массы и размеров побега.

Фенотипирование корней. Корневая система высших растений играет ключевую роль в поставке питательных веществ из почвы, при этом ее архитектура может сильно различаться как между разными видами, так и у различных генетических форм одного вида. Основные морфологические характеристики корней можно разделить на макроскопические и микроскопические. К макроскопическим признакам относятся длина главного корня, его диаметр, количество боковых корней, разветвленность и наличие придаточных корней. Микроскопические признаки включают количество, размер и расположение корневых волосков, а также характеристики эпидермальной ткани. Изучение сложной структуры корневой системы неразрушающими методами представляет определенные трудности, поэтому многие исследования проводятся в контролируемых лабораторных условиях. При этом растения, как правило, выращиваются на гидропонике [35]. Один из типичных подходов для наблюдения за ростом корней - использование прозрачной агаризованной среды и выращивание растений в вертикально расположенных чашках. Корни растут по поверхности среды и доступны для непрерывной неинвазивной двумерной визуализации.

Фенотипирование семян и плодов. Системы фенотипирования позволяют автоматически оценивать форму, размеры и другие морфологические характеристики различных видов семян. Для фенотипирования плодов используются различные методы автоматизированного анализа изображений, полученных с помощью лидаров (LiDAR, от англ.: Light Detection and Ranging).

К методам анализа изображений можно отнести:

- Методы цветового анализа: позволяют определить диапазон цветов на изображении и выделить определенные цвета или цветовые группы.
- Методы текстурного анализа: позволяют определить текстуру изображения и выделить области с различными текстурными характеристиками.
- Методы сегментации изображений: позволяют разделить изображение на отдельные объекты или области в соответствии с их признаками (цвет, яркость, текстура и т.д.).
- Методы фильтрации изображений: применяются для удаления шумов и улучшения качества изображения путем применения различных фильтров (размытие, усиление контраста и др.).

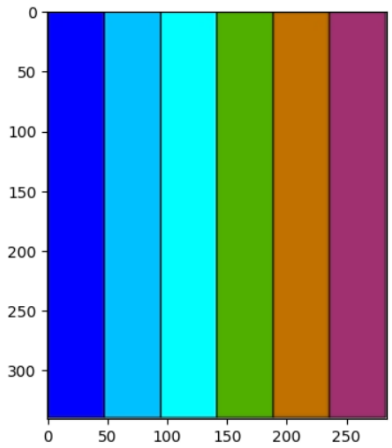
- Методы детекции объектов: позволяют обнаружить на изображении объекты определенного типа или класса (лица, автомобили, здания и др.).
- Методы распознавания образов: позволяют определить класс объекта на изображении (распознавание лиц, цифр, букв и т.д.).
- Методы извлечения признаков: позволяют выделить характеристики объектов на изображении для их последующей классификации или распознавания.
- Методы анализа движения: позволяют отслеживать движущиеся объекты на изображении, определять их траектории и скорость движения.
- Методы машинного обучения и нейронных сетей: используются для автоматизации анализа изображений, обучения моделей распознавания и классификации объектов на изображениях.

Предобработка изображений является важным шагом в научном анализе изображений, так как позволяет улучшить качество данных и подготовить изображения для последующего анализа. Вот некоторые основные шаги предобработки изображений для научного анализа: приведение к одному цветовому каналу, если изображение имеет несколько цветовых каналов (RGB, CMYK и т. д.); удаление шума, применяя фильтры (например, медианный или Гауссов) для удаления шума с изображения и улучшения его качества; улучшение контраста, используя методы улучшения контраста (например, эквализация гистограммы) для улучшения четкости изображения; размер изображения, изменяя размер изображения на необходимый для получения правильных результатов при анализе; определение границ, применяя методы обнаружения границ (например, оператор Собеля или детектор краев Кэнни) для выделения границ объектов на изображении; бинаризация, преобразуя изображение в черно-белое, чтобы упростить анализ (методы бинаризации, например, пороговая бинаризация); удаление ненужных деталей, например, фон или шум с помощью методов сегментации (например, разбиение на регионы или морфологические операции); нормализация интенсивности пикселей изображения для корректного сравнения и анализа; сегментация изображений на отдельные объекты и структуры для дальнейшего анализа и извлечения характеристик.

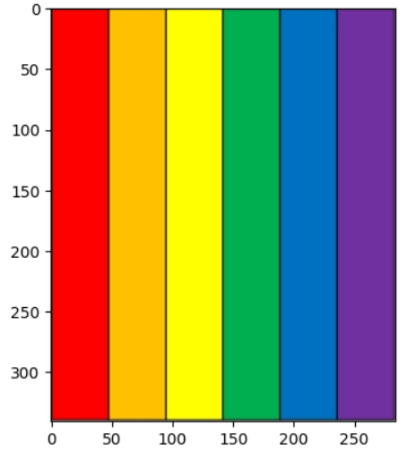
После выполнения всех этих шагов изображение будет готово для дальнейшей работы, такой как распознавание образов, классификация объектов, извлечение признаков и многое другое. Рассмотрим некоторые из этих операций с помощью модулю OpenCV-python.

```
#приведение к одному цветовому каналу
import cv2
from matplotlib import pyplot as plt

#загрузка изображения по дефолту в BGR
image = cv2.imread('image.jpg')
```

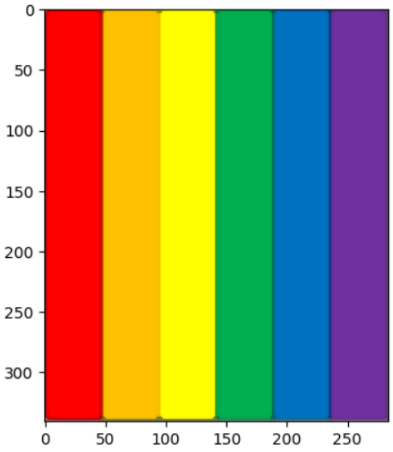
```
#отображение изображения
plt.imshow(image)
plt.show()
```




```
#преобразование изображения в серый цветовой канал
image_RGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

#отображение изображения
plt.imshow(image_RGB)
plt.show()
```

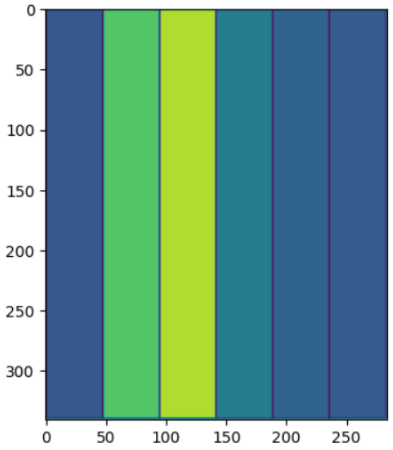


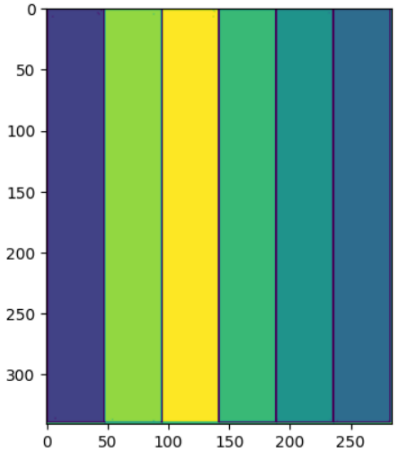

```
# удаления шума (за шум будем принимать чёрные полосы)
#применение медианного фильтра
filtered_image = cv2.medianBlur(image_RGB, 5)

plt.imshow(filtered_image)
plt.show()
```



#гистограммное выравнивание
#загрузка изображения в оттенках серого
image = cv2.imread('image.jpg', 0)
equ = cv2.equalizeHist(image)

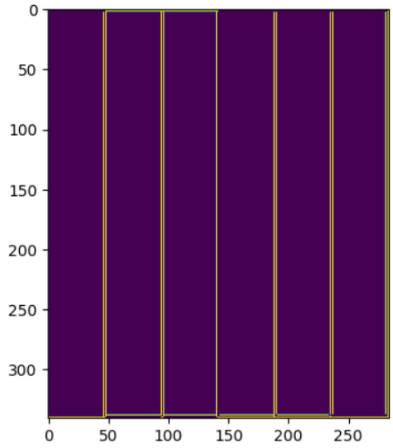
plt.imshow(image)
plt.show()
plt.imshow(equ)
plt.show()





#нахождение границ
#применение оператора Canny
edges = cv2.Canny(image, 15, 30)

plt.imshow(edges)
plt.show()



1.8. Общие понятие в Machine Learning

Вопросы:
1. Общие понятие в Machine Learning.
2. Применение методов Machine Learning.
3. Моделирование и метрики.

Machine learning – это подраздел искусственного интеллекта, который относится к разработке алгоритмов и моделей, которые позволяют компьютерным системам извлекать знания из данных и принимать самостоятельные решения на их основе.

Данные – основа любого алгоритма машинного обучения. Данные могут быть структурированными (таблицы, базы данных) или неструктурированными (тексты, изображения).

Обучающий набор данных – это набор примеров, на котором модель обучается. Он состоит из входных данных (признаки) и выходных данных (метки или целевые переменные).

Модель – это математическое представление проблемы или задачи, которое строится на основе обучающих данных. Модель пытается найти зависимости или закономерности в данных.

Обучение – процесс, в котором модель «обучается» на обучающем наборе данных, чтобы найти оптимальные параметры или взвешивания для решения задачи.

Тестирование – процесс, в котором модель проверяется на тестовом или валидационном наборе данных, чтобы оценить ее производительность и эффективность.

Переобучение и недообучение – две проблемы, с которыми может столкнуться модель машинного обучения. Переобучение происходит, когда модель слишком хорошо подстраивается под обучающие данные и не может обобщить новые данные. Недообучение происходит, когда модель слишком проста и неспособна корректно решить задачу.

В основном многие конкретные задачи решают с помощью достаточно точных алгоритмов, которые заскриптованы на определённые условия, однако такие алгоритмы не могут дать правильного решения на задачу, которая выбивается из прописанных условий. Из-за низкой универсальности возможно решать только конкретные задачи конкретными инструкциями. И если подобной программе на вход подаётся объект, который не подходит к инструкции, то и код заложенный в программе выполняться не будет.

Это проблема весьма актуальна для биологических данных, так как они по своей сути уникальные, большие и при этом имеют сложную структуру. Из этой проблемы вытекает следующая — невозможность корректной работы с большими массивами сложной информации, такой как геномы или феномы организмов.

Big Data – это термин, используемый для описания огромного по объему набора данных, который со временем растет экспоненциально. Такие данные настолько большие и сложные, что ни один из традиционных инструментов управления данными не может их эффективно хранить или обрабатывать. Можно отметить, что для больших данных характерно правило «четырёх V», а именно: Volume (объём) – большой объём; Variety (разнообразие) – низкоструктурированные и разнородные; Velocity (быстрота) – требуют быстрой обработки; Value (ценность) – позволяют получить значимые результаты анализа данных.

Data Mining – это процесс обнаружения в больших данных ранее неизвестных нетривиальных и практически полезных закономерностей. Основу методов Data Mining составляют всевозможные методы классификации, моделирования и прогнозирования, основанные на применении деревьев решений, искусственных нейронных сетей, генетических алгоритмов, эволюционного программирования, ассоциативной памяти, нечёткой логики. К методам Data Mining нередко относят статистические методы (дескриптивный анализ, корреляционный и регрессионный анализ, факторный анализ, дисперсионный анализ, компонентный анализ, дискриминантный анализ, анализ временных рядов, анализ выживаемости, анализ связей). Одно из важнейших назначений методов Data Mining состоит в наглядном представлении результатов вычислений (визуализация), что позволяет использовать инструментарий Data Mining людьми, не имеющими специальной математической подготовки. Среди классов задач Data Mining можно выделить следующие: Классификация; кластеризация; прогнозирование; поиск ассоциаций; поиск последовательностей.

Классификация – определение класса объекта по его заранее известным характеристикам. Методы решения: k-ближайшего соседа (k-Nearest Neighbor); байесовские сети (Bayesian Networks); деревья решений; нейронные сети (neural networks) и т.п.

Кластеризация – выявление некоторых признаков объектов и группировка этих объектов в кластеры. Методы решения: графовые алгоритмы;

вероятностные алгоритмы кластеризации; иерархические алгоритмы кластеризации; алгоритм k-means; сдвиг среднего значения; спектральная кластеризация; DBSCAN.

Тут важно пояснить отличия классификации от кластеризации.

- Цель: Классификация используется для присвоения объектов к определенным заранее известным классам, в то время как кластеризация используется для разделения объектов на группы на основе их сходства.
- Учитывание меток: В классификации требуется иметь заранее известные метки классов, по которым проводится разделение, в то время как в кластеризации метки не учитываются и группы формируются на основе внутренней структуры данных.
- Методы использования: Классификация применяется для обучения моделей машинного обучения с учителем, тогда как кластеризация используется в методах без учителя.
- Результаты: В классификации результатом является присвоение объектов к определенным классам, в то время как результатом кластеризации является разделение объектов на гнезда, основанные на их сходстве.
- Интерпретация: Результаты классификации обычно легко интерпретируемы, поскольку классы заранее известны, в то время как результаты кластеризации могут быть сложны для интерпретации из-за отсутствия четко определенных классов.

Прогнозирование – построение модели поведения процесса по накопленным данным и их особенностях при помощи методов математической статистики, нейронных сетей и тд. Для прогнозирования в Data Mining часто используются различные алгоритмы, такие как линейная и логистическая регрессия, случайные леса, градиентный бустинг и нейронные сети. Для создания точных прогнозов необходимо провести анализ данных, подготовить их, выбрать наиболее подходящий алгоритм и провести обучение модели на исторических данных.

Поиск ассоциаций – нахождение частых зависимостей между объектами, где найденные зависимости представляются в виде правил. Для поиска ассоциаций в данных используются различные алгоритмы, такие как Apriori и FP-Growth. Эти алгоритмы анализируют частоту встречаемости комбинаций элементов данных и находят те, которые встречаются вместе чаще, чем ожидалось бы случайно. Поиск ассоциаций имеет широкий спектр применений, от рекомендательных систем в интернет-магазинах до анализа клиентской базы и выявления скрытых связей. Он является важным инструментом для понимания поведения пользователей и принятия бизнес-решений на основе данных.

Поиск последовательностей (шаблонов/паттернов) – нахождение временных закономерностей между цепью событий. Для поиска последовательностей в данных используются различные алгоритмы, такие как алгоритм Apriori для анализа часто встречающихся комбинаций элементов, алгоритм GSP (Generalized Sequential Pattern) для поиска общих

последовательностей, алгоритм SPAM для нахождения часто встречающихся подпоследовательностей и другие.

Последовательностей можно искать не только в числовых данных, но и в тексте, изображениях и видео, что делает эту технику очень универсальной и полезной для анализа различных типов данных. Результаты поиска последовательностей могут помочь выявить скрытые закономерности и паттерны в данных, что, в свою очередь, может быть использовано для прогнозирования поведения, принятия решений и оптимизации процессов в различных областях.

Поиск ассоциаций и поиск последовательностей являются двумя разными методами анализа данных. Основной задачей поиска ассоциаций является выявление связей между различными элементами в наборе данных. Поиск ассоциаций используется для выявления тесно связанных элементов, которые часто встречаются вместе. Примером задачи поиска ассоциаций является анализ покупок в магазине для определения часто встречающихся комбинаций товаров. Одним из популярных алгоритмов для поиска ассоциаций является алгоритм Apriori.

Основной задачей поиска последовательностей является изучение порядка следования элементов в наборе данных. Поиск последовательностей используется для выявления шаблонов последовательности элементов в данных. Примером задачи поиска последовательностей является анализ поведения пользователей на веб-сайте для определения последовательности действий. Одним из популярных алгоритмов для поиска последовательностей является алгоритм GSP (Generalized Sequential Pattern).

Таким образом, основное отличие между поиском ассоциаций и поиском последовательностей заключается в том, что первый метод выявляет связи между различными элементами, а второй - анализирует порядок следования элементов в данных.

Полученные алгоритмы используются для определения структур макромолекул, а также их функций, с целью объяснения различных биологических явлений. В области фармацевтики методы Data Mining также имеют достаточно широкое применение.

Это задачи исследования эффективности клинического применения определенных препаратов, определение групп препаратов, которые будут эффективны для конкретных групп пациентов, для аннотации геномов, в приложениях клеточной биологии, в гистологии и медицине при идентификации тканей.

Разберём более подробно методы/алгоритмы машинного обучения. Остановимся далеко не на всех, но зацепим самые ключевые, а именно дерево решений, Random forest, логистическая регрессия, метод ближайшего соседа, нейронные сети.

Прежде всего, необходимо сказать о разделении выборки на тренировочную и тестовую/валидационную. Это необходимо для того, чтобы модель обучилась на одном наборе данных (тренировочном), а проверить её

можно было на наборе, который она не встречала ранее, так как велик шанс переобучения модели и следствие заучивание конкретных данных и невозможность предсказания новых. Для разделения можно использовать функцию train_test_split из scikit-learn. Предположим, у нас есть два списка, один из них содержит признаки экземпляра, второй метку класса, к которому принадлежит экземпляр:

```
data_f = [1,0,1,2,4,6,4,2,4,6,7,8,4,3,4,6,6,3,4,8,0]
data_t = [0,0,0,0,1,1,1,0,1,1,1,1,1,0,1,1,1,0,1,1,0]

#импорт функции для разделения выборки
from sklearn.model_selection import train_test_split

#в аргументах передаются признаки, таргетная переменная и размер тестовой выборки
train_x, test_x, train_y, test_y = train_test_split(data_f, data_t, test_size=0.3)

print(train_x, test_x, train_y, test_y)

#([2, 4, 6, 3, 1, 6, 0, 4, 4, 2, 7, 4, 8, 4],
[1, 3, 6, 0, 4, 6, 8],
[0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1],
[0, 0, 1, 0, 1, 1, 1])
```

Как видно из вывода кода выборки разделились в процентном соотношении 70/30, при этом это было сделано со списком признаков и списком таргетов. Также стоит сказать пару слов про метрику accuracy, далее мы рассмотрим более подробно другие метрики. Accuracy – количество правильных прогнозов / количество всех прогнозов * 100%. Допустим имеются два списка с прогнозами модели и реальными ответами из тестового набора.

```
true_l = [1,1,0,1,1,0]
pred_l = [1,1,0,1,0,1]

#импорт accuracy
from sklearn.metrics import accuracy_score

print(accuracy_score(true_l, pred_l))

#точность модели 66%
# 0.6666666666666666
```

Дерево решений – это графическая модель принятия решений в форме дерева, где каждый узел представляет собой альтернативное решение, а каждое ребро – возможный результат этого решения.

Деревья решений используются в машинном обучении для классификации и прогнозирования. Они позволяют анализировать данные и делать выводы о прогнозируемых значениях. Процесс построения дерева решений начинается с корня и последовательно принимает решения на основе определенных критериев, разделяя данные на каждом узле, пока не будет достигнуто конечное решение. Дерево решений может быть использовано для принятия решений в различных областях, таких как медицина, финансы, бизнес и т.д. Одним из преимуществ деревьев решений является их легкость интерпретации и возможность представления результатов в понятной форме. Для базового использования дерева решений в Python можно использовать библиотеку scikit-learn.

```
#импорт дерева решений
from sklearn.tree import DecisionTreeClassifier

#задаём переменной класс дерева решений
model = DecisionTreeClassifier()

#далее код будет одинаковый для всех моделей
#метод fit – обучает модель на тренировочном наборе
model.fit(x_train, y_train)

#метод predict показывает ответы модели для каждого семпла
predicted_test = model.predict(x_test)

print(accuracy_score(y_test, predicted_test))

#0.9298245614035088
```

Случайный лес (Random forest) – это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

- Выбирается подвыборка обучающей выборки размера samplesize и по ней строится дерево (для каждого дерева – своя подвыборка);
- Для каждого дерева выбирается свое число случайных признаков (для каждого нового расщепления – свои случайные признаки);
- Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают высоту дерева, число

объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

#импорт модели случайного леса
from sklearn.ensemble import RandomForestClassifier

Метод ближайших соседей (k-nearest neighbors algorithm, k-NN) - Метод k-ближайших соседей используется для решения задачи классификации. Он относит объекты к классу, которому принадлежит большинство из k его ближайших соседей в многомерном пространстве признаков. Число k – это количество соседних объектов в пространстве признаков, которые сравниваются с классифицируемым объектом. Иными словами, если k=10, то каждый объект сравнивается с 10-ю соседями.

Например, есть два объекта, описанных десятью признаками. Нужно определить, к какому объекту – A или B – ближе объект X. Для этого необходимо рассчитать евклидово расстояние между объектами A/B и X и определить, какой объект ближе.

#импорт модели ближайшего соседа
from sklearn.neighbors import KNeighborsClassifier

Логистическая регрессия схожа с линейной, за исключением того факта, что вместо вычисления значения y, она оценивает к какой категории принадлежит точка данных. Логистическая регрессия хорошо подходит для решения задач бинарной классификации – назначая разным категориям значения 0 и 1 соответственно.

По опыту построения предыдущих моделей, постройте модель логистической регрессии. Посчитайте confusion matrix, accuracy, precision и recall. Тюнингуя параметр C (от 0.05 до 5 с шагом 0.1), найдите наилучшую точность на тесте. C (C Regularization Parameter) – это параметр штрафования или неверной классификации. За каждую ошибку модель штрафуется, что может увеличить эффективность, но и увеличить переобучение.

#импорт модели логистической регрессии
from sklearn.linear_model import LogisticRegression

Искусственная нейронная сеть – это математическая модель, созданная по подобию биологических нейронных сетей. На входной слой нейронов поступает некая информация, которая по синапсам переходит на следующий слой. При этом каждый синапс обладает собственным коэффициентным весом, а любой следующий нейрон в новом слое может иметь несколько входов. Информация передается дальше до тех пор, пока не дойдет до конечного выхода.

Модель узнает, какие связи между нейронами важны для успешного прогнозирования во время обучения. На каждом этапе тренировки сеть использует математическую функцию, чтобы определить, насколько точным был ее последний прогноз по сравнению с ожидаемым.

Эта функция генерирует серию значений ошибок, которые могут использоваться системой для расчета того, как модель должна обновлять значение весов, прикрепленных к каждой ссылке, с конечной целью повышения точности прогнозов сети.

```
#импорт модели нейронной сети
from sklearn.neural_network import MLPClassifier
```

Что касается валидации моделей, мы обсудили одну метрику, accuracy, однако она не является исчерпывающей, так как сильно зависит от сбалансированности классов. Рассмотрим матрицу ошибок, матрица ошибок – это таблица, которая позволяет визуализировать эффективность алгоритма путем сравнения прогнозируемого значения целевой переменной с ее фактическим значением. Столбцы матрицы представляют наблюдения в прогнозируемом классе, а строки – наблюдения в фактическом классе. Матрица ошибок представлена на рисунке 11.

	Класс 0 / Negative (pred_label)	Класс 1 / Positive (pred_label)
Класс 0 / Negative (true_label)	TN (True Negative) (Кол-во элементов класса **0**, **верно** предсказанных как класс **0**)	FP (False Positive) (Кол-во элементов класса **0**, **неверно** предсказанных как класс **1**)
Класс 1 / Positive (true_label)	FN (False Negative) (Кол-во элементов класса **1**, **неверно** предсказанных как класс **0**)	TN (True Positive) (Кол-во элементов класса **1**, **верно** предсказанных как класс **1**)

Рисунок 11 – Матрица ошибок

Precision и recall – это две важные метрики, используемые в оценке качества работы алгоритмов машинного обучения и классификации.

Precision (точность) – это доля правильно предсказанных положительных классов среди всех объектов, которые модель предсказала как положительные. Формула для precision:

(1)

$$Precision = TP / (TP + FP)$$

где TP (True Positives) – количество правильно предсказанных положительных классов, а FP (False Positives) - количество неправильно предсказанных положительных классов.

Recall (полнота) – это доля правильно предсказанных положительных классов среди всех объектов, которые принадлежат к положительному классу в истинной выборке. Формула для recall:

(2)

$$Recall = TP / (TP + FN)$$

где FN (False Negatives) – количество неправильно предсказанных отрицательных классов.

Обе метрики имеют значения в диапазоне от 0 до 1, где более высокие значения указывают на лучшее качество работы алгоритма. Они взаимосвязаны и помогают оценить баланс между точностью и полнотой в классификации.

F1 score – это метрика, которая используется для оценки качества моделей машинного обучения. Она является гармоническим средним между точностью (precision) и полнотой (recall) и представляет собой отношение их среднего геометрического к их среднему арифметическому. F1 score позволяет учесть как ошибки false positives, так и false negatives, обеспечивая более обобщенную оценку производительности модели. Чем выше значение F1 Score, тем лучше качество модели. Формула F1 score:

(3)

$$F1\ score = 2 * (precision * recall) / (precision + recall)$$

AUC-ROC (Area Under the Receiver Operating Characteristic curve) - это метрика, используемая для оценки качества модели классификации. Она измеряет площадь под кривой ROC (Receiver Operating Characteristic), которая показывает отношение между долей верных положительных результатов (True Positive Rate) и долей ложных положительных результатов (False Positive Rate) при различных порогах классификации.

Чем ближе значение AUC-ROC к 1, тем лучше модель классификации. Значение 0.5 означает случайное угадывание, а значение ближе к 0 обычно означает, что модель хуже случайного угадывания. AUC-ROC имеет преимущество перед другими метриками, такими как точность и полнота, потому что она учитывает все пороги классификации, а не только один. Пример ROC кривой на рисунке 12.

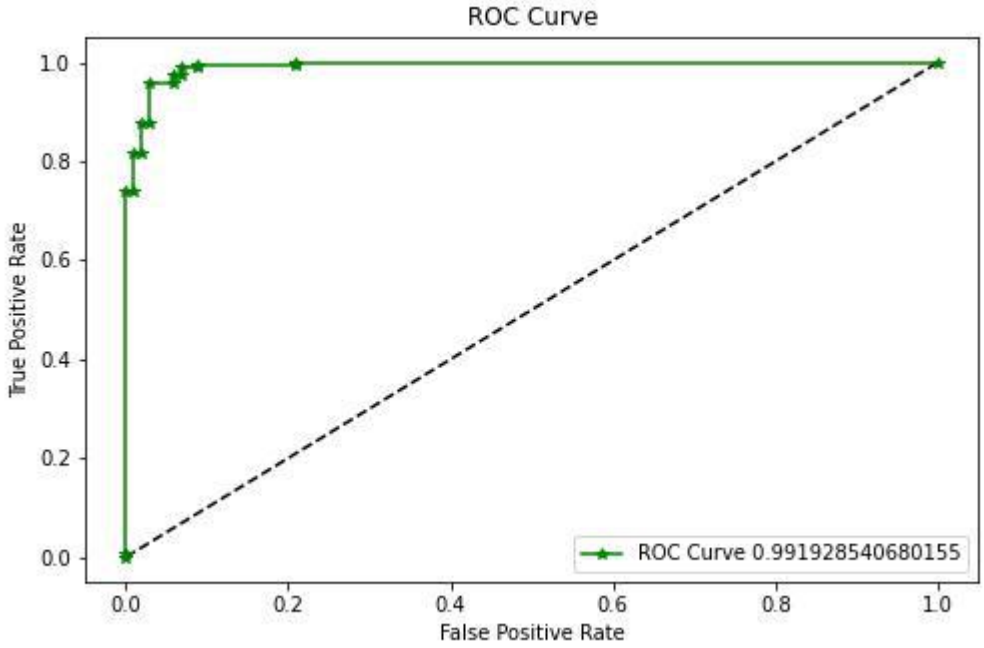


Рисунок 12 – Пример ROC кривой

Эти метрики позволяют оценить качество моделей классификации и регрессии в машинном обучении и выбрать наилучшую модель для конкретной задачи.

1.9. Построение и применение нейронных сетей в биологии

Вопросы:
1. Устройство нейронных сетей.
2. Обучение нейронных сетей.
3. Проверка моделей на переобучение.

Нейронные сети находят широкое применение в биологии, так как они могут обрабатывать большие объемы данных и проводить сложные анализы, что помогает в понимании биологических процессов и развитии новых методов исследований. Например, нейронные сети могут использоваться для анализа геномных данных, предсказания структуры белков, классификации клеточных изображений, идентификации генов-маркеров и многое другое.

Использование нейронных сетей для биологических задач становится все более популярным и распространенным. Например, нейронные сети используются для анализа генетических данных, предсказания структуры белков, диагностики заболеваний на ранних стадиях, анализа медицинских изображений (например, для детектирования рака на рентгенограммах), предсказания реакции организма на лекарственные препараты и многое другое.

Одним из примеров использования нейронных сетей в биологии является прогнозирование взаимодействий между молекулами, что может помочь в разработке новых лекарственных препаратов. Также нейросети могут

использоваться для анализа генетических данных и выявления связей между генами и различными болезнями, что может помочь в разработке персонализированной медицины.

Нейронные сети позволяют обрабатывать и анализировать большие объемы данных и выявлять закономерности, которые могут быть незаметны при традиционных методах анализа. Таким образом, использование нейронных сетей в биологических задачах может значительно улучшить точность диагностики, прогнозирование и лечение различных заболеваний.

Нейронная сеть – последовательность нейронов, соединённых между собой связями. В простом представлении нейрон – это объект, который имеет в себе численное значение обычно от 0 до 1 или от -1 до 1. Нейрон принимает на вход информацию и передаёт её дальше. Между нейронами имеется связь, основной и единственной характеристикой этой связи является вес. Нейрон в искусственной нейронной сети обычно состоит из следующих основных элементов: входной сигнал, веса, функция активации, выходной слой.

Входные сигналы: нейрон принимает входные сигналы от других нейронов или от внешнего источника. Веса: каждый входной сигнал умножается на соответствующий вес, который определяет важность этого сигнала для нейрона. Функция активации: суммированные значения проходят через функцию активации, которая определяет, будет ли нейрон активирован или нет. Функция активации определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения. Она проверяет произведенное нейроном значение Y на предмет того, должны ли внешние связи рассматривать этот нейрон как активированный, или его можно игнорировать. На рисунке 13 представлены популярные функции активации.

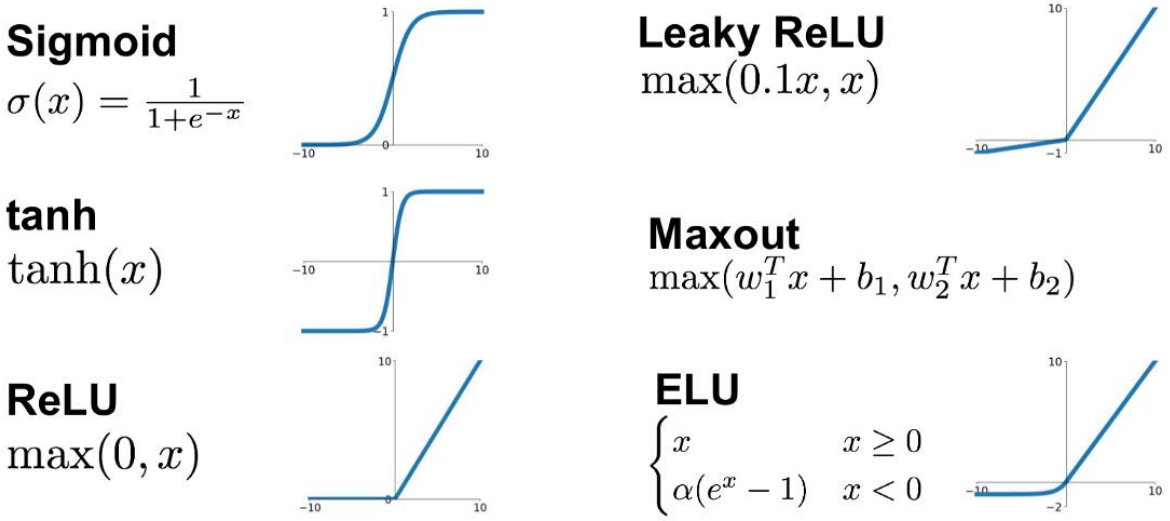


Рисунок 13 – Примеры функций активации

Для большей гибкости и устойчивости нейросети используют узел смещения (b). С помощью узлов смещения можно изменять количество входных и скрытых нейронов. Более простой способ понять, что такое смещение: в стандартном уравнении для прямой линии: y = ax + bВыходной

сигнал: результат работы функции активации служит выходным сигналом нейрона.

Таким образом, нейрон в искусственной нейронной сети является простым вычислительным элементом, который получает входные сигналы, обрабатывает их и выдает выходной сигнал в зависимости от входных данных и параметров весов.

Обучение нейронной сети – поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется в необходимый выходной, с ожиданием от сети способности обобщать какие-то признаки и решать задачу на различных входных данных.

Выборки можно разделить на обучающую и тестовую выборку. На обучающей выборке нейронная сеть обучается. После того, как сеть выдаст корректный результат для входных данных, сеть проверяют на тренировочной выборке, которую она прежде не видела, и которая отличается от обучающей.

Обучение с учителем – система обучения при которой нейронная сеть обучается на парах входное значение «стимул» – выходное значение «реакция».

В обучении без учителя у модели есть набор данных, и нет явных указаний, что с ним делать. Нейронная сеть пытается самостоятельно найти корелляции в данных, извлекая полезные признаки и анализируя их.

Обучение с частичным привлечением учителя характеризуется своим названием: обучающий датасет содержит как размеченные, так и неразмеченные данные.

Обучение с подкреплением – этот тип машинного обучения требует использования системы вознаграждения/штрафа. Цель – вознаградить машину, когда она учится правильно, и наказать машину, когда она учится неправильно.

Процесс обучения – обратное распространение ошибки. Обучение алгоритмом обратного распространения ошибки предполагает два прохода по всем слоям сети: прямого и обратного.

Прямой – от входного к выходному сигнал. В результате генерируется набор выходных сигналов и всех синаптических весов, которые фиксируются. Во время обратного прохода все веса настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в обратном направлении. Далее частная производная ошибки вычисляется по каждому весу (эти частные дифференциалы отражают вклад каждого веса в общую ошибку). Веса настраиваются с целью максимального приближения выходного сигнала сети к желаемому.

TensorFlow и PyTorch – это две из самых популярных библиотек машинного обучения и глубокого обучения. Обе библиотеки предоставляют различные инструменты и функциональность для создания и обучения нейронных сетей, а также инструменты для визуализации данных и оценки моделей.

TensorFlow был разработан в Google, и имеет широкую поддержку в индустрии и академическом сообществе. Он предоставляет гибкие инструменты для создания и обучения различных типов нейронных сетей, включая сверточные нейронные сети, рекуррентные нейронные сети и генеративно-состязательные сети. TensorFlow также имеет поддержку для распределенного обучения на нескольких устройствах, что делает его отличным выбором для масштабируемых проектов.

PyTorch, с другой стороны, разработан в Facebook и имеет более удобный и интуитивно понятный интерфейс, чем TensorFlow. Он использует динамический граф вычислений, что делает его более гибким и удобным для экспериментов и исследований. PyTorch также имеет богатую библиотеку модулей и оптимизаторов, что облегчает создание и обучение сложных нейронных сетей.

В целом, выбор между TensorFlow и PyTorch будет зависеть от потребностей и предпочтений. Если нужна широкая поддержка и разнообразие инструментов, то TensorFlow может быть лучшим выбором. Если важен более интуитивный интерфейс, то PyTorch может быть более подходящим вариантом. Пример простой нейронной сети на Python с использованием Tensorflow:

```
import tensorflow as tf

# Создаем нейронную сеть
model = tf.keras.models.Sequential([
   tf.keras.layers.Flatten(input_shape=(28, 28)),  # Преобразование в
одномерный массив
   tf.keras.layers.Dense(64, activation='relu'),
   tf.keras.layers.Dense(10, activation='softmax')
])

# Загружаем данные
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Нормализуем данные
x_train, x_test = x_train / 255.0, x_test / 255.0

# Компилируем модель
model.compile(optimizer='adam',
          loss='sparse_categorical_crossentropy',
          metrics=['accuracy'])

# Обучаем модель
model.fit(x_train, y_train, epochs=5)
```

```
# Оцениваем точность модели
print(model.evaluate(x_test, y_test))

#[0.09587574750185013, 0.9710000157356262]
```

Пример простой нейронной сети на Python с использованием PyTorch:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision

# Создаем нейронную сеть
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x), dim=1)
        return x

# Загружаем данные
mnist = torchvision.datasets.MNIST
transform =
torchvision.transforms.Compose([torchvision.transforms.ToTensor()])
train_data = mnist(root='./data', train=True, transform=transform,
download=True)
test_data = mnist(root='./data', train=False, transform=transform,
download=True)

# Создаем загрузчики данных
train_loader = torch.utils.data.DataLoader(dataset=train_data, batch_size=64,
shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_data, batch_size=64,
shuffle=False)

# Обучаем модель
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
for epoch in range(5):
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images.view(-1, 784))
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Оцениваем точность модели
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images.view(-1, 784))
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy: {} %'.format(100 * correct / total))

# Accuracy: 95.43 %
```

Переобучение – это ситуация, когда модель слишком хорошо подстроена под тренировочные данные и теряет способность обобщать на новых данных. Для проверки моделей на переобучение можно использовать следующие методы:

- Разделение данных на тренировочный и тестовый наборы. Подавая модели на обучение только тренировочные данные и проверяя ее точность на тестовом наборе данных, можно оценить, насколько модель обобщает данные.
- Кросс-валидация. Этот метод позволяет повторять процесс разделения данных на тренировочный и тестовый наборы несколько раз, что позволяет получить более стабильную оценку качества модели.
- Использование регуляризации. Добавление штрафов за сложность модели (например, L1 или L2 регуляризация) помогает предотвратить переобучение, уменьшая веса признаков модели.
- Построение кривой обучения. Построение графика зависимости ошибки на тренировочном и тестовом наборах от размера обучающей выборки позволяет оценить, насколько модель переобучена или недообучена.
- Использование ансамблей моделей. Сочетание нескольких моделей (например, случайного леса или градиентного бустинга) может помочь уменьшить переобучение за счет различных подходов и методов обучения.

Проведение этих методов поможет оценить модели на переобучение и выбрать наилучший вариант для конкретной задачи.

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

Примерная тематика практических занятий

Лабораторное занятие № 1. Работа классами и виртуальной средой.

Лабораторное занятие № 2. BioPython и его возможности.

Лабораторное занятие № 3. Алгоритмы выравнивания. Глобальное и локальное выравнивание.

Лабораторное занятие № 4. Работа с протеомными данными.

Лабораторное занятие № 5. Модели машинного обучения.

Лабораторное занятие № 6. Устройство нейронных сетей.

Лабораторное занятие № 7. Функции активации и построение нейронных сетей.

Лабораторное занятие № 8. Использование специализированных библиотек для построения нейронных сетей.

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. Перечень контрольных мероприятий управляемой самостоятельной работы студентов

Формой текущей аттестации по дисциплине «Программирование на Python в биологии» учебным планом предусмотрен зачет.

Для оценки профессиональных компетенций студентов используется следующий диагностический инструментарий:

- защита отчета о выполненной лабораторной работе;
- контрольная работа.

Оценка за ответы на лекциях (коллоквиумы) и защиту лабораторных занятий может включать в себя полноту ответа, наличие аргументов, примеров из практики и т.д.

К сдаче зачёта по учебной дисциплине допускаются студенты:

- не имеющие пропусков по лабораторным занятиям. В случае пропуска лабораторного занятий отработка проводится в виде написания реферата по теме пропущенного занятия с последующей его защитой;
- сданными контрольными работами на отметку выше или равной минимальной зачётной (4 и выше).

Формирование отметки за текущую успеваемость:

- отчёт по лабораторной работе - 50 %;
- контрольная работа - 50 %.

3.2. Примерный перечень заданий для управляемой самостоятельной работы студентов

Управляемая самостоятельная работа (консультационно-методическая поддержка и контроль) осуществляется преимущественно в дистанционной форме и обеспечивается средствами образовательного портала БГУ LMS Moodle (https://edubio.bsu.by/course/view.php?id=1078). Объем часов на составление и размещение заданий, консультации и контроль, осуществляемые с использованием технологий дистанционного обучения, планируется в пределах учебных часов, отведенных на УСР.

Тема 2.1 Алгоритмы для анализа геномных и протеомных данных

(2 ч/ДО). 1. Классы в Python. Задачи геномики. Основные компоненты ПЦР. Трансляция. Глобальное выравнивание. Локальное выравнивание. Преобразование Барроуза-Уилера. Цели и задачи протеомики. Количественный и качественный анализ протеома. Методы получения пространственной структуры белковых молекул.

Форма контроля: контрольная работа на образовательном портале LMS Moodle.

Тема 2.2 Машинное обучение и анализ изображений

(2 ч/ДО). Цели и задачи анализа изображений. Методы анализа имиджей. Предобработка имиджей. Общие понятие в Machine Learning. Применение методов Machine Learning. Моделирование и метрики. Устройство нейронных сетей. Обучение нейронных сетей. Проверка моделей на переобучение.

Форма контроля: контрольная работа на образовательном портале LMS Moodle.

3.3. Примерный перечень вопросов к зачету

1. Цели и задачи биоинформатики. Прикладная область информатики.
2. Структура ДНК, РНК. Азотистые основания.
3. Основные этапы репликации.
4. Основные этапы транскрипции.
5. Основные этапы трансляции.
6. Форматы хранения информации о последовательности аминокислот.
7. Метод скользящего окна. Алгоритмы выравнивания.
8. Глобальные и локальные выравнивания. Марковские цепи в молекулярной генетике.
9. Сравнение последовательностей биологических полимеров. Алгоритм Бойера-Мура. Реализация алгоритма. Пример работы алгоритма.
10. Упорядоченные структуры для индексации. Хеш-таблицы для индексации. Геномные индексы, используемые в исследованиях.
11. Принцип Pigeonhole. Примеры моделей в геномике.
12. Структура белков. Количественный и качественный анализ протеома. Классификация подходов к идентификации белков.
13. Масс-спектрометрия. Анализ масс-спектров пептидов с помощью программного обеспечения.
14. Методы определения пространственной структуры белков. Рентгеноструктурный анализ.
15. Методы биоинформатики для построения пространственной структуры белка.
16. Цели и задачи анализа изображений.
17. Способы получения феномных данных (RGB/флуоресцентный/термальный имиджинг).
18. Методы анализа имиджей. Предобработка имиджей.
19. Machine Learning. Общие понятие в Machine Learning.
20. Методы. Data Mining. Применение методов Machine Learning.
21. Моделирование и метрики. Деревья решений и регрессии.
22. Использование нейронных сетей для биологических задач. Устройство нейронных сетей.
23. Обучение нейронных сетей. Библиотеки TenserFlow и PyTorch.
24. Анализ эффективности моделей машинного обучения. Проверка моделей на переобучение. Проверка эффективности.

4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1. Рекомендуемая литература

Перечень основной литературы:

1. Стефанов, Василий Евгеньевич. Биоинформатика: учебник для студентов высших учебных заведений, обучающихся по техническим и естественнонаучным направлениям / В.Е. Стефанов, А.А. Тулуб, Г.Р. Мавропуло-Столяренко. – Москва: Юрайт, 2022. – 252 с.

2. Леек, Артур. Введение в биоинформатику / А. Леек; пер. с англ. А.А. Миронова, В.К. Швядоса. – 2-е изд. – Москва: БИНОМ. Лаборатория знаний, 2015. – 318 с.

3. Кребс, Джоселин. Гены по Лыоину / Дж. Кребс, Э. Голдштейн, С. Килпатрик; пер. с англ. [И.А. Кофиади и др.] под редакцией Д.В. Ребрикова и Н.Ю. Усман. – 5-е изд., испр. – Москва: Лаборатория знаний, 2022. – 919 с

Перечень дополнительной литературы:

1. Антао, Тьяго. Биоинформатика с Python. Книга рецептов. / Т. Антао: пер. с англ. И.Л. Люско Москва: ДМК Пресс, 2023, – 344 с.

2. Стивенсон, Бен. Python. Сборник упражнений / Б. Стивенсон; пер. с англ. А. Ю. Гинько. – Москва: ДМК Пресс, 2021. – 238 с.

3. Stevens, Tim Г, Boucher, Wayne. Python Programming for Biology: Bioinformatics and Beyond / T.J. Stevens, W. Boucher. – Cambridge University Press, 2015. – P. 711.

4.2. Электронные ресурсы

1. Учебная программа учреждения высшего образования по учебной дисциплине «Учебная программа учреждения высшего образования по учебной дисциплине специальности 6-05-0511-05 Биоинженерия и биоинформатика. Электронная библиотека БГУ [Электронный ресурс]. – Режим доступа: https://elib.bsu.by/handle/123456789/296999. – Дата доступа: 03.05.2023.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Andreas D. Baxevanis, B. F. Francis Ouellette, Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins, 2nd Edition, 2018. – P. 504.
2. Arthur M. Introduction to Bioinformatics. 2008. – P. 1423.
3. Choudhuri S. Bioinformatics for Beginners: Genes, Genomes, Molecular Evolution, Databases and Analytical Tools. 2014. – P. 225.
4. Buffalo V. Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools. 2015. – P. 508.
5. Russakovsky O., et al. ImageNet Large Scale Visual Recognition Challenge. // International Journal of Computer Vision. – Vol. 115. – 2015. – P. 211–252.
6. Bengio Y., et al. Deep Learning and Neural Networks. // Nature. – Vol. 521. – 2015. – P. 436–444.
7. LeCun Y. et al. Deep Learning. // Nature. – Vol. 521. – 2015. – P. 436–444.