Bioinformatics Resources

Lecture & Exercises

Prof. B. Rost, Dr. L. Richter,
Institut für Informatik I12
Bioinformatics Resources

- Organization
- Schedule
- Overview
Organization

- **Lecture:** Friday 9-12, i.e. 9.30-11.45 o’clock
  10 - 15 min break in between
  Room 01.09.034

- **Exercise:** Monday 14-16 o’clock room tba, starting Mon, May 6\textsuperscript{th}
  OR
  Friday 13-15 o’clock room 01.09.014
  starting Fri, May 10\textsuperscript{th}
  -> Doodle?
Team Behind the Course

BioinfRes SoSe 19 / 4
Preliminary Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 26th</td>
<td>Intro, General Overview (1. sh.)</td>
<td>June 14th</td>
<td>NoSql 2 (7.sh.)</td>
</tr>
<tr>
<td>May 3rd</td>
<td>Sequence Databases (2. sh.)</td>
<td>June 21th</td>
<td>To be discussed w students</td>
</tr>
<tr>
<td>May 10th</td>
<td>Sequence Databases (3. sh.)</td>
<td>June 28th</td>
<td>MongoDB, JavaScript (8.sh.)</td>
</tr>
<tr>
<td>May 17th</td>
<td>Structure Databases (4. sh.)*</td>
<td>July 5th</td>
<td>Node.js Applications (9.sh.)*</td>
</tr>
<tr>
<td>May 24th</td>
<td>SQL (5. sh.)</td>
<td>July 12th</td>
<td>PredictProtein</td>
</tr>
<tr>
<td>May 31st</td>
<td>To be discussed w students</td>
<td>July 19th</td>
<td>Wrap Up, Q&amp;A</td>
</tr>
<tr>
<td>June 7th</td>
<td>SQL, NoSql (6. sh)</td>
<td>(July 26th</td>
<td>(LMU Statistics) or) July 31st Exam</td>
</tr>
</tbody>
</table>

* These exercises can earn you a bonus
Schedule Details

- If there is no lecture the exercise takes place one week later
- Exercise sheets are published on Fridays and discussed Fri/Mon the week after
- Last sheet/exercise: Jul 12th, Fri/Mon 19th/22nd
- Exam: (working date): July (26th or) 31st, to be discussed with the audience, room yet tbd
Overview

- Fifth/last iteration
- Experience driven syllabus – subject to change
- Depending on the progress in the lecture single topics could be added or dropped in comparison to other years
- The sequence of topics might be shuffled
- Hybrid nature: presentation of existing resources are blended with back- and front-end technology
Exercises

- Exercises help to convert knowledge into a skill
- Practical application of topics covered in the lecture
- Active exploration of bioinformatics resources
- Implementing various aspects of a bioinformatics resource
- Use Python/Biopython as common platform
- Explore visualizations with JavaScript
Terms and Meanings

- What does “resource” actually means?
- A Google query about “Bioinformatics Resource” yields about 40 Mio hits
- Falls roughly into three categories:
  - databases
  - tools
  - service centers
Working on a Definition

- A collection of information which is useful to do research in the area of life sciences/computational biology
- Contains the information itself
- Provides appropriate interfaces to access the information
- (May provide) Tools for interactive data analysis
Genbank* / NCBI

- NIH genetic sequence database
- Annotated collection of all publicly available DNA sequences
- Part of the International Nucleotide Database Collaboration together with DNA DataBank of Japan (DDBJ) and the European Molecular Biology Laboratory (EMBL)
- Genbank flat file format

Genbank IIc

- New release every 2 months
- Retrievable via FTP from the NCBI website
- Current release is 231, April, 2019
- 321,680,566,570 bases, from 212,775,414 reported sequences in Genbank; 4,421,986,382,065 from 993,732,214 sequences in WGS, April 2019
Genbank IIb

- New release every 2 months
- Retrievable via FTP from the NCBI website
- Current release is 224.0, February, 2018
- 253,630,708,098 bases, from 207,040,555 reported sequences in Genbank;
  2,608,532,210,351 from 564,286,852 sequences in WGS, Feb 2018
Genbank IIa

- (231,824,951,552 bases from 200,877,884 sequences in Genbank; 2,035,032,639,80 from 451,840,147 sequences in WGS, Apr 2017)
- (211,423,912,047 bases from 191,739,511 reported sequences, Apr 2016)
- (187,893,826,750 bases from 181,336,445 reported sequences, Feb 2015)
Genbank III

- Three main divisions: CoreNucleotide, dbEST, dbGSS
- Querying over Entrez Nucleotide
- Interactive BLAST analysis with user sequences
- Programmatic access via NCBI e-utilities
Swissprot*

- Official name: UniProtKB/Swiss-Prot
- Current release: 2019_03
- 559634 sequence entries, comprising 201129965 amino acids abstracted from 265241 references
  - (557012 sequence entries comprising 199,714,119 residues from 258,792 references)
  - (554,241 (Apr. 2016:548,208) sequence entries comprising 198,410,167 residues from 251,858 references)
  - (550960 sequence entries, 195,282,524 residues abstracted from 235893 references last year

Swissprot/Uniprot II

- history
- manual annotation process
- standard operation procedure
- controlled vocabularies
- guidelines
- offered services: BLAST, Align, ID mapping
- associated services

BioinfRes SoSe 19 / 17
Other Uniprot Services

- TrEMBL
- Proteomes
- UniRef
- UniParc
- programmatic access
PDB*

- History
- 140,3393 structures, incl. 126,694 proteins by now
  - (129,367 structures, incl. 120,137 proteins in 2018)
  - (118,087 structures, incl. 115,169 proteins in 2016)
  - (108124 structures, incl. 100450 proteins in 2015)
- PDB formats
- Data upload/validation
- Data dictionaries

*http://www.rcsb.org/pdb/statistics/holdings.do
PDB II

- Retrieval
- Programmatic access
- Visualization with the different views
- File format transitions: pdb and mmcif
SCOP/e

- Structural Classification of Proteins
- History, current version is SCOPe 2.05
- Changes in SCOPe
- Access
- Needed/recommended additional software
PFAM

- current version is 32.0, September 2018
- 17929 entries
- what is is about
- categories
- interactive use
- programmatic access
Prosite

- Prosite
  - current version 2019_03 of Apr 10\textsuperscript{th} 2019 contains 1829 documentation entries, 1310 patterns, 1240 profiles and 1264 ProRule.
  - UniRule format and ProRule
  - access
  - typical use and interfaces
PubMed and discussion forums

- What is it for
- Search opportunities
- Linking to other information sources
- Search strategies
- (A tour through various discussion forums)
File Formats*

- Genbank/EMBL
- SWISSPROT
- PDB: mmCIF
- (High Throughput NGS data:)
  - BAM, SAM
  - VCF

* mostly integrated
File Formats

- Equivalence and transformations between different formats
- XML formats
- (RDF formats)
SQL

- What is it all about?
- SQL basics
- Data types
- Table creation and manipulation
- Join
- Select
SQL II

- Keys
- Indexes
- Performance influence of indexes
- Similarity search vs substrings
- Permissions
SQL III

- Transactions
- Setup, administration, backup
- Programmatic access
- mySQL, postgresQL
SQL IV

- General hints for database design
- Do’s and don’ts
- Normalization ultra light
No SQL

- Definitions of NoSQL
- Advantages / disadvantages
- Underlying theory
- Typical use cases
- Types of No-SQL database
- Query (languages)
No SQL Systems

- MongoDB
- (CouchDB)
- Neo4J
- Programmatic access
(Storing Facts)*

- Triple stores
- Data model
- RDF refresher
- Query language: sparql
- Examples

* optional, might be dropped
Programming Libraries

- Roadshow of programming libraries dedicated to bioinformatics:
  - (bioperl)
  - biopython
  - bioJS/D3
  - Visualization
Graphical User Interfaces*

- Interactive user interfaces with JavaScript
- Language basics
- Programming model
- Client/server communication with json
* to be confirmed
JavaScript

- Libraries for data visualization/bioinformatics
- matplotlib
- bioJS
- D3
Web Services I

- Types of web services
- Web service components
- Integration of web services in software
Selected Bioinformatics Suites

- presented in dedicated sessions:
  - Aquaria
  - PredictProtein
Summary I

- Aim of this module:
  - Shape the concept of a bioinformatics resource
  - Become familiar with some of the most prominent examples out there
  - Get in touch with the underlying technology
  - Collect ideas and experience how to realize a new bioinformatics resource
Summary II

- Hands on (interaction) experience extends existing knowledge/skills
- Backend technology, i.e. various database models
- Frontend technology to realize the UI/ design rationales
- Communication models
Grading:

- Graded by a written exam 90 min
- Scheduled day July 31st depends on:
  - available room
  - number of participants
- Exam admission: no admission limit
- With sufficient performance in the two marked exercises you can earn a bonus, sufficient -> tbd
- The bonus applies only if you pass the exam
Exercises

- Exploration of available resources
- Simple to intermediate programming tasks
- Publication of the task in week x
- Presentation of solutions in week x+1
Exercises II

- 9 -10 exercise sheets
- Work in groups of 2 for the bonus *(deprecated)*
- Discussion with the audience
Exercises III

- (Groups fixed for the bonus)
- New sheets are published on Friday
- Submission is due on Friday morning for all groups
- One slot for exercises
Questions & Answers
Programming Exercises

- We will use Python for our programming exercises
- Scripting language
- Basic understanding of Python should be sufficient to understand the presented code snippets
- Vivid community for support and development
Programming Exercises II

- Object oriented
- Good integration with database systems and web access
- Good integration with sophisticated data analysis tools like: numPy, sciPy, matplotlib
- BioPython
- Interactive notes: jupyter and notebooks
- Spyder
Structure your research work

- Computational biology is data driven
- Results matter -> more results matter more
- Other than e.g. software development there is no final release version and all prior bugs/version are abandoned
- Appropriate documentation of the experiments to reconstruct the intermediate steps is important, otherwise you may end up with result01 - result1000 files
Our recommended Software Setup

- Anaconda
- iPython/Jupyter notebooks
Anaconda

- Python distribution ([https://www.continuum.io](https://www.continuum.io))
- Clever package manager: conda
- Allows a complete installation including various configurations next to each other in the user space
- No privileges needed
- Your host system is not modified
- Works with Windows, OS X, Linux (no excuse aka “I don’t have a Linux installation...”)

BioinfRes SoSe 19 / 55
Some snippets from the conda cheat sheet

- Use "conda create –n xxx biopython" to create a new environment xxx AND install biopython
- Use "(source) activate xxx" to activate this environment in your shell
- Allows different versions of python to be installed at the same time!
iPython/Jupyter

- [http://jupyter.org](http://jupyter.org)
- Supports many different languages, we use it for python
- Use conda to install the package (if not installed): conda install jupyter
- Easy start of notebook: jupyter notebook
Advantages of a Notebook

- Allows you a seamless integration of:
  - (rich) text
  - (live) code
  - (live) results
  - visualizations
- Tie together your analysis scripts, the results and an interpretation/discussion
- You can archive and share the notebooks easily
More Details about Notebooks

- Supports and records interactive analysis session
- Variables and values are persistent between cells
- Provides full power of Data Science tools via imported modules: Bio, Pandas, Numpy, Scipy
- Good for data processing: Pandas
- Good statistics support: Numpy, Scipy
- Good visualization support: Matplotlib
Biopython

- Advantage of easier / more clear syntax than Perl
- Oriented to BioPerl
- Supports a lot of common bioinformatics file formats
- Supports access to online services like NCBI, Expasy ...
- More interfaces for bioinformatics software
  
  [http://biopython.org/DIST/docs/tutorial/Tutorial.html](http://biopython.org/DIST/docs/tutorial/Tutorial.html)
Biopython

- [http://biopython.org](http://biopython.org)

- If installed: “import Bio” loads it in your scripts


```python
from Bio.Seq import Seq
# create a sequence object
my_seq = Seq('CATGTAGACTAG')

# print out some details about it
print 'seq %s is %i bases long' % (my_seq, len(my_seq))
print 'reverse complement is %s' % my_seq.reverse_complement()
print 'protein translation is %s' % my_seq.translate()
```
Biopython

seq CATGTAGACTAG is 12 bases long
reverse complement is CTAGTCTACATG
protein translation is HVD*

taken from http://biopython.org/wiki/SeqIO:
from Bio import SeqIO
handle = open("example.fasta", "rU")
for record in SeqIO.parse(handle, "fasta") :
    print record.id
handle.close()

from Bio import SeqIO
record = SeqIO.read(open("single.fasta"), "fasta")
Dedicated Data Structures

- sequence (Seq): beside the sequence of residues it allows also to provide an Alphabet object -> kind of type safety for DNA and protein sequences
- Typical functions like `complement()`, `reverse_complement()`
Dedicated Data Structures

- Parsing functions for different sequence formats
- Parsing functions for alignment formats know about the different components
- As well as respective output functions
- Different translation tables
- Various predefined alphabets
Python Basics (Version 3)

- [https://docs.python.org/3/tutorial/](https://docs.python.org/3/tutorial/)
- [https://py-tutorial-de.readthedocs.io/de/python-3.3/](https://py-tutorial-de.readthedocs.io/de/python-3.3/)

- Good interactive handling, i.e. you can evolve and evaluate your code directly in a python shell
- Later you can include it in your script
Interactive Python-Shell / Prompt

- start the interpreter without argument
- obey indentation rules
- all definitions from the same session are available
- end block with two newlines (one empty line)
General Language Features

- Interpreted but often based on C-libraries in the backend -> quite fast
- Uses white space instead of {} to define syntactic units / blocks
- object oriented
- dynamically (strongly) typed language => no type declarations for variables BUT always implicitly typed with the initialization/assignment
Data Types (not complete)

- Basic data types:
  - numerical types comparable to Perl, C, Java
  - strings: ‘…’, or “…”
  - boolean: True, False

- Complex collection types

- Functions are first class members i.e. you can assign a function to a variable and you can return a function from another function call
Functions

- declaration with key word `def`
- quite versatile parameter lists (see tutorial)
- implicit or explicit return value
- `def myFunc():`
  ```python
  print("Hello world")
  ```
- `myFunc` <- function object
- `myFunc()` <- function call/execution
Functions As Return Values

def genPower(power):
    def specPower(base):
        return base ** power
    return specPower

square = genPower(2)
cube = genPower(3)
square(5)  # 25
cube(5)    # 125
Important Syntax

- White space (tabs, spaces) and : are used to structure the code in blocks, similar to {} in other languages
- Same indentation == same block
- Blocks of if/for/while etc. statements are always indented
- No semicolons needed
- Parentheses often not mandatory
- Be careful with line breaks!
Loops

- **while**: needs a boolean condition
- **for**: needs a list to be iterated
- **use** `range(x)` **to generate** the list 0-x for x iterations
- **break** and **continue** analogous to C or Java

```python
for w in words:
    print (w, len(w))

# if you want to iterate by numbers you
# have to use range()
for i in range(len(a)):
    print i, a[i]
```
Functions and Arguments

- Definition of functions:

```python
def fib(n):    # write Fibonacci series up to n
    """Print a Fibonacci series up to n. """
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b
```

- Arguments can be passed by name or position
- Arguments can have default values -> optional in the call
- Default values are evaluated at the point of definition
Functions and Arguments

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")

parrot(1000)                          # 1 positional argument
parrot(voltage=1000)                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM') # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000) # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump') # 3 positional
    # arguments
parrot('a thousand', state='pushing up the daisies')  # 1
    # positional, 1 keyword
taken from https://docs.python.org/3/tutorial/controlflow.html
Functions and Arguments

- positional arguments have to be given always before named ones

- even more arguments are possible:
  - *anyName: a tupel containing additional positional arguments
  - **anyDifferentName: a dictionary containing additional keyword arguments
  - *anyName must occur before **anyDifferentName

- if you need a stub doing nothing: pass
Useful Data Types

- **Sequence types:**
  - Supports easy check for an element
  - Mutable types: List, Bytearray
  - Immutable: String, Tuple

- **Slicing:** act on sub sets not only on single elements
Other Collection Types

- Set: every element exists only once
- Dictionary:
  - can store key/value pairs
  - key has to be immutable (hashable)
- derived from Dictionary:
  - DefaultDictionary
  - Counter
List Comprehension

- very powerful
- compact notation of looping of lists
- needs:
  - substrate list
  - function/operator
- yields:
  - derived list
Example for List Comprehension

```python
r = range(10)
>>> print(r)
range(0, 10)
>>> [a for a in r]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> [a*2 for a in r]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> [a*2 for a in r if a%2 == 0]
[0, 4, 8, 12, 16]
```
Slicing

- [] operator for lists
- allows ranges:
  - [from .... to[
- a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- a[1:5]
- [1, 2, 3, 4]
- analogous for Dictionaries and string lists