

Faculdade de Engenharia da Universidade do Porto

Data Enrichment for Data Mining Applied to Bioinformatics and Cheminformatics Domains

Luís Ricardo Marques Oliveira



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Rui Camacho

September 20th, 2021

Data Enrichment for Data Mining Applied to Bioinformatics and Cheminformatics Domains

Luís Ricardo Marques Oliveira

Mestrado Integrado em Engenharia Informática e Computação

September 20th, 2021

Abstract

Increasingly more complex problems are being addressed in life sciences. Acquiring all the data that may be related to the problem in question is paramount. Equally important is to know how the data is related to each other and to the problem itself. On the other hand, there are large amounts of data and information available on the Web. Researchers are already using Data Mining and Machine Learning as valuable tools in their investigations, albeit the usual procedure is to look for the information based on the inductive models.

So far, despite the great successes already achieved with Data Mining and Machine Learning, it is not easy to integrate this vast amount of available information in the inductive process with propositional algorithms. The algorithms of propositional machine learning are very dependent on data attributes. It still is hard to identify which attributes are more suitable for a particular task in the research. It is also hard to extract relevant information from the enormous quantity of data available. We aggregate the data available and derive features that ILP algorithms can use to induce descriptions and solve problems.

We have created a web platform to help solve relevant bioinformatics (particularly Genomics) and Cheminformatics problems. It fetches information about compounds, substances, genes and proteins from public genomics, protein and chemical repositories. The data enrichment aggregates material from different sources, produces a conversion dictionary and generates Prolog facts from all the information collected. With the facts, Prolog systems use inductive logic programming to induce rules and solve specific Bioinformatics and Cheminformatics case studies.

Our main motivation was to address the problem of integrating domain information into the inductive process of propositional Data Mining and Machine Learning techniques by enriching the training data to be used in inductive logic programming systems. The solution successfully addresses the problem at hand and is very easy and simple to use. It has already been used to generate three Prolog knowledge base datasets with millions of clauses each.

Keywords: Genomics, Cheminformatics, Data Mining, Classification, Enrichment, ILP

Resumo

Problemas cada vez mais complexos estão a ser tratados na área das ciências da vida. A aquisição de todos os dados que possam estar relacionados com o problema em questão é primordial. Igualmente importante é saber como os dados estão relacionados uns com os outros e com o próprio problema. Por outro lado, existem grandes quantidades de dados e informações disponíveis na Web. Os investigadores já estão a utilizar Data Mining e Machine Learning como ferramentas valiosas nas suas investigações, embora o procedimento habitual seja procurar a informação baseada nos modelos indutivos.

Até agora, apesar dos grandes sucessos já alcançados com a utilização de Data Mining e Machine Learning, não é fácil integrar esta vasta quantidade de informação disponível no processo indutivo, com algoritmos proposicionais. Os algoritmos proposicionais de Machine Learning são muito dependentes dos atributos dos dados. Ainda é difícil identificar quais os atributos mais adequados para uma determinada tarefa na investigação. É também difícil extrair informação relevante da enorme quantidade de dados disponíveis. Agregamos os dados disponíveis e derivamos características que os algoritmos de ILP podem utilizar para induzir descrições, resolvendo os problemas.

Criamos uma plataforma web para ajudar a resolver importantes problemas de Bioinformática (particularmente Genómica) e Quimioinformática. Vai buscar os dados a repositórios públicos de dados genómicos, proteicos e químicos. O enriquecimento dos dados vai juntar material de diferentes fontes, produzir um dicionário de conversão e gerar factos Prolog a partir de toda a informação reunida. Com os factos, sistemas Prolog utilizam programação lógica indutiva para induzir regras e resolver casos específicos de Bioinformática e Cheminformática.

A nossa principal motivação era abordar o problema da integração de informação de domínio no processo indutivo de técnicas proposicionais de Data Mining e Machine Learning, por enriquecimento dos dados de treino a serem utilizados em sistemas de programação de lógica indutiva. A solução resolve com sucesso o problema em questão e é muito fácil e simples de usar. Já foi usada para gerar três datasets de Base de Conhecimentos Prolog, com mais de dois milhões de cláusulas cada.

Keywords: Genómica, Quimioinformática, Data Mining, Classificação, Enriquecimento, ILP

Acknowledgements

I wish to thank Professor Rui Camacho, for his guidance, help and attention throughout this project.

A special thank you to Marlene for her support and for encouraging me to take a five-year (integrated) master's course at the age of 42...

Luís Marques Oliveira

“Being in a minority, even in a minority of one, did not make you mad. There was truth and there was untruth, and if you clung to the truth even against the whole world, you were not mad.

George Orwell, *Nineteen Eighty-Four*

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Objectives	2
1.4	Dissertation structure	2
2	Bioinformatics and Cheminformatics	3
2.1	Repositories and their API	15
2.2	Tools	31
2.3	Related Work	32
2.4	Summary	33
3	Data Mining and Machine Learning Background	35
3.1	Relational Data Mining and Machine Learning	35
3.2	Inductive Logic Programming	57
3.3	Tools	63
3.4	Summary	67
4	Implementation	69
4.1	Requirements	69
4.2	Technologies	71
4.3	Boilerplate	75
4.4	Architecture	80
4.5	User Stories	87
4.6	User Interface	90
4.7	Requests & Repositories	98
4.8	Deployment	105
4.9	Summary	107
5	Case Studies	109
5.1	Handling Proteins	109
5.2	Handling Chemistry	112
5.3	Handling Genes	114
5.4	Summary	117
6	Conclusions and Future Work	119
6.1	Conclusions	119
6.2	Future Work	120
	References	121

A Databases	125
B More interface pages examples	133

List of Figures

2.1	Bioinformatics applications	4
2.2	Omics from the genome to the phenome	7
2.3	Cheminformatics viewed by Frank K. Brown	8
2.4	QSAR / QSPR flow diagram	9
2.5	Cheminformatics basic steps	9
2.6	Chemical structure representations	10
2.7	Cangen algorithm for SMILES representation	11
2.8	InChIKey generation workflow diagram.	12
3.1	CRISP-DM diagram	36
3.2	RapidMiner Linear Regression output example	37
3.3	Examples of knowledge representation	38
3.4	Cluster representations	39
3.5	Cluster representations	39
3.6	K-nearest neighbours example	41
3.7	Comparison of Bayes networks classifiers	42
3.8	Random forest diagram	43
3.9	Support Vector Machines diagram	44
3.10	Diagram of a generic perceptron	46
3.11	Diagram of a Artificial Neural Network	47
3.12	Precision vs Accuracy	48
3.13	Extended confusion matrix	49
3.14	Confusion matrix	49
3.15	ROC and AUC	51
3.16	Hierarchical agglomerative and divisive approaches	54
3.17	Density-based neighbourhoods	55
3.18	Sets with positive and negative examples	57
4.1	MVC architecture diagram	72
4.2	Django application architecture diagram	73
4.3	Application Architecture	80
4.4	Navigation Flow	90
4.5	Landing Page	91
4.6	Repositories page	92
4.7	Kegg input page	94
4.8	Bulk input page	95
4.9	PaDEL Page	96
4.10	.csv to .arff pipeline	97
4.11	Cheminformatics conversion table in Administration Area	97

5.1	Proteomics case study pipeline	111
5.2	Chemistry case study pipeline	113
5.3	Genomics case study pipeline	115
A.1	UML of the conceptual data model of the PDB database	125
A.2	UML of the conceptual data model of the Pubchem database	126
A.3	UML of the conceptual data model of the Kegg database	127
A.4	UML of the conceptual data model of the Genbank database	128
A.5	UML of the conceptual data model of the Ensembl database	129
A.6	UML of the conceptual data model of the Go database	130
A.7	UML of the conceptual data model of the Converter database	130
A.8	UML of the conceptual data model of the Knowledge Base database	131
B.1	Genomic field page	133
B.2	Proteomic field page	134
B.3	Bulk Input page with help tooltip	135
B.4	Outputs page	136
B.5	CSV to ARFF output page	137
B.6	Output Prolog page	138
B.7	PDB input page	139
B.8	PDB search results	140
B.9	PDB output page	141
B.10	Pubchem repo page for compounds (2D output)	142
B.11	Pubchem repo page for compounds (3D output)	143
B.12	Pubchem repo page for substances	144
B.13	Pubchem Output cut	145
B.14	Kegg search results	146
B.15	Genbank repo page	147
B.16	Genbank repo page with help tooltip	148
B.17	Genbank search results	149
B.18	Genbank output page	150
B.19	Ensembl repo page	151
B.20	Ensembl Gene output page	152
B.21	Ensembl Protein output page	152
B.22	Orphan Exon not saved	153
B.23	Login page	154
B.24	Change Password page	154
B.25	Database export example	155
B.26	Add New user page	155
B.27	Edit user page	156

List of Tables

2.1	IUPAC's one- and three-letter amino acid codes	6
2.2	Table of Cheminformatics methods	14
2.3	NCBI GenBank RefSeq accession number prefixes and molecule types. .	18
4.1	Functional Requirements	70
4.2	Restrictions	70
4.3	Technical Requirements	71
4.4	Frontend endpoints	81
4.5	Backend endpoints	84
4.6	Main User stories, their usages and corresponding UI Figures	88
4.7	Repositories corresponding UI Figures	89
4.8	User management User stories, their usages and corresponding UI Figures	89
4.9	Properties obtainable from Pubchem	99
4.10	Kegg entity types, examples and regular expressions	101

Abbreviations

ADMET	Absorption, Distribution, Metabolism, and Excretion - Toxicity
ANN	Artificial Neural Network
ANSI	American National Standards Institute
API	Application Product Interface
ARFF	Attribute-Relation File Format
AUC	Area Under the Curve
BLAST	Basic Local Alignment Search Tool
CCDS	Consensus Coding Sequence
CID	(Pubchem) Compound IDentifier
CDK	Chemistry Development Kit
CNN	Convolutional Neural Network
CRISP-DM	Cross-industry Standard Process for Data Mining
CSV	Comma Sparated Values
DAG	Directed Acyclic Graph
DBSCAN	Density Based Spatial Clustering of Application with Noise
DM	Data Mining
DNA	DeoxyriboNucleic Acid
DTD	Document Type Definition
DSSTox	Distributed Structure-Searchable Toxicity
EBI	The European Bioinformatics Institute
EMBL	European Molecular Biology Laboratory
EPA	(United States) Environmental Protection Agency
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
GAF	Gene Annotation File
GAN	Generative Artificial Network
GNN	Graph Neural Network
GUI	Graphical User Interface
HIV	Human Immunodeficiency Virus
HTTP	HyperText Transfer Protocol
ID3	Iterative Dichotomiser 3
IDE	Integrated Development Environment
ILP	Inductive Logic Programming
InChi	IUPAC International Chemical Identifier
ISO	International Organization for Standardization
IUPAC	International Union of Pure and Applied Chemistry
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KEGG	Kyoto Encyclopedia of Genes and Genomes

KNN	K-Nearest Neighbours
MDL	Meaning Definition Language
MIM	Mendelian Inheritance in Man
ML	Machine Learning
mRNA	Messenger RiboNucleic Acid
MTV	Model-Template-Controller
MVC	Model-View-Controller
NCBI	National Center for Biotechnology Information
NCI	National Cancer Institute
NIH	National Institutes of Health
npm	Node Package Manager
OMIM	Online Mendelian Inheritance in Man
OWL	Web Ontology Language
PCR	Polymerase Chain Reaction
pip	Pip Installs Packages
PIR	Protein Information Resource
PDB	Protein Data Bank
QSAR	Quantitative Structure-Activity Relationship
QSPR	Quantitative Structure-Property Relationship
RCSB	Research Collaboratory for Structural Bioinformatics
RDBMS	Relational Database Management System
ROC	Receiver Operating Characteristics
RNA	RiboNucleic Acid
RNN	Recurrent Neural Networks
SAR	Structure-Activity Relationship
rRNA	Ribosomal RiboNucleic Acid
SIB	Swiss Institute of Bioinformatics
SDF	Structural Data Files
SID	(Pubchem) Substance IDentifier
SMILES	Simplified Molecular-Input Line-Entry System
SSN	Shared Nearest Neighbours
SQL	Structured Query Language
STS	Sequence Tagged Site
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
tRNA	Transfer RiboNucleic Acid
UML	Unified Modeling Language
URL	Uniform Resource Locator
WEKA	Waikato Environment for Knowledge Analysis
WWW	World Wide Web
XML	eXtensible Markup Language

Chapter 1

Introduction

The evolution of the Internet and the creation of collaborative databases allowed access to increasing amounts of relevant data to the most diverse fields.

Data Mining and Machine Learning techniques are increasingly being used both to take advantage of the available information and to enable the tracking of volumes of data impossible to scrutinise by humans. They are being used mainly to gain knowledge and to make predictions.

In the field of life sciences, there has been a great evolution in what is called Bioinformatics and Cheminformatics, in sequencing, analysis and prediction.

In this chapter, we will contextualise this dissertation, the motivation behind it, the objectives we aim to achieve, the methodology used and the structure.

1.1 Context

So far, despite the great successes already achieved with the use of Data Mining and Machine Learning, is very difficult to integrate this vast amount of available information in the inductive process, with propositional algorithms.

The algorithms of propositional machine learning are very dependent on data attributes. It still is hard to identify which attributes are more suitable for a particular task in the research. It is also hard to extract relevant information from the enormous quantity of data available.

1.2 Motivation

Our main motivation is to address the problem of integrating domain information into the inductive process of propositional Data Mining and Machine Learning techniques, through the enrichment of the training data to be used in inductive logic programming systems.

The use of domain knowledge Machine Learning algorithms can build simpler models but with high accuracy, build more intelligible models that can be amenable

to validation by experts and it can also provide new and valuable knowledge in the domain under analysis that would otherwise pass unnoticed.

And how can domain knowledge be integrated into the process of propositional induction? Through pre-processing, we can do it directly in the form of an attribute, transform a set of values into a new, more powerful set with feature construction - such as using the sub-graph of a recursive molecule as a new attribute - or, without pre-processing, use a relational algorithm. For that we have Inductive Logic Programming.

We will concentrate the available data, derive features that can be used by ILP algorithms to induce descriptions that can solve the Bioinformatics and Cheminformatics problems.

1.3 Objectives

We aim to address the problem at hand and use the enrichment information in the induction process instead of the usual procedure of looking for the information based on the induced models, as biologists do today.

Our objectives are creating a web platform that allows obtaining, in appropriate repositories, using API, relevant information for problems in the area of Bioinformatics, particularly Genomics, and of Cheminformatics, as well. The impact of this data enrichment on Bioinformatics and Cheminformatics's concrete applications will also be evaluated, using specific case studies that we will also choose.

We expect to carry out enough experiments with the implemented solution that can be conclusive and prove the enrichment was fruitful, enabling ILP systems to outperform the propositional approach on certain situations.

1.4 Dissertation structure

The dissertation is divided into two main parts: one is the state of the art and background both in data mining and technologies, and the other is the solution and its architecture as well as the results obtained.

In the first part, in Chapter 2, we will describe and understand both Bioinformatics and Cheminformatics. What they are, what they deal with and how. In some cases, we will discuss certain Data Mining and Machine Learning techniques that are used. Then, in Chapter 3, we will take a look at Data Mining and Machine Learning approaches, schemes and algorithms and how they address specific problems. We will dedicate a special section to discuss inductive logic programming.

In the complementing part, in Chapter 4, we will present the different technologies chosen for the development of this work, its implementation, architecture and uses. In Chapter 5 is described how we have handled the three case studies, the flow of execution and how to get outputs needed to the data mining processes.

Before ending, in Chapter 6, we will discuss the results and finish with a conclusion.

Chapter 2

Bioinformatics and Cheminformatics

With the technological and computer advancements, many fields of expertise in life sciences increasingly adopted new computer-based solutions to solve old and new challenges. In biology and chemistry, these eventually evolved to become areas on their own and stabilised their names as Bioinformatics and Chem(o)informatics.

Bioinformatics

The origin of Bioinformatics lay in the path that led to DNA sequencing. It goes back to 1953 when Watson and Crick published the DNA structure, and the 1960s, with the accumulation of data and knowledge of biochemistry and protein structure. The systematisation of knowledge of the three-dimensional protein structure, in 1984 gave Margaret O. Dayhoff the title of mother of bioinformatics [11].

The human genome project and the improvement of therapeutic processes with the identification of a new gene or mutation into a known gene are some of the visible aspects of what bioinformatics is about. It usually refers to the use of computational tools in the study of biological problems and issues, also covering human health-related applications such as planning new drugs through protein structures from nucleotide sequences [55].

In 2001, Luscombe et al. defined bioinformatics as

“The application of computational techniques to understand and organise the information associated with biological macromolecules.” [31]

Bioinformatics main applications are shown in Figure 2.1. We can divide them into two main areas: sequence prediction or manipulation and 3D structure prediction, even though the sequence determines the structure [55].

Sequencing involves the following applications[55]:

- comparisons between sequences (alignment);
- identification of patterns in sequences (signatures);
- characterisation of evolutionary relationships (phylogeny);

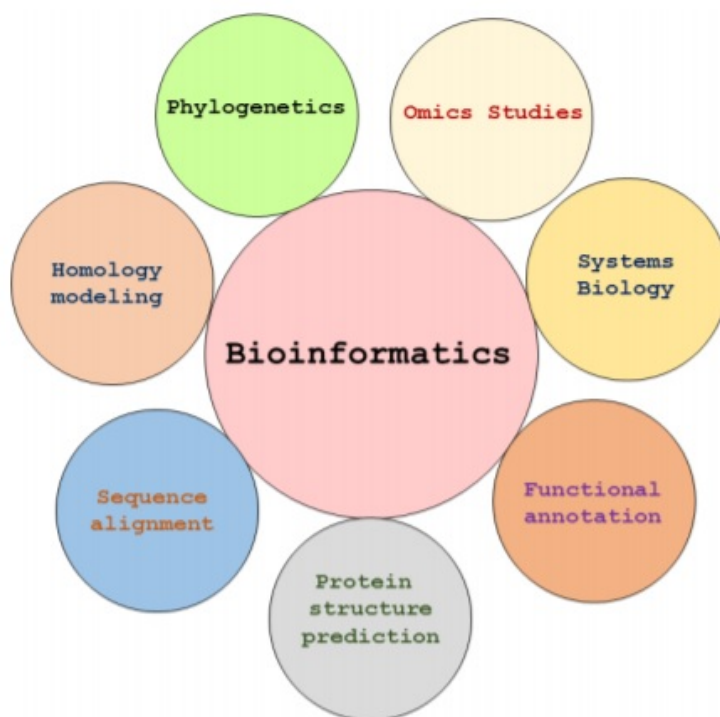


Figure 2.1: Bioinformatics applications. Source: [11]

- construction and annotation of genomes (genomics);
- network building (biology of systems).

and 3D structure prediction, these[55]:

- obtaining 3D models for proteins and other biomolecules (comparative modelling);
- identification of modes of interaction between molecules (proteomics);
- selection of compounds with higher inhibition potential (proteomics);
- characterisation of molecular flexibility (molecular dynamics);
- evaluation of the effect of changes in structure and molecular environment in the dynamics and function of biomolecules (molecular dynamics).

Gupta, in [18], lists several areas of application in bioinformatics: molecular medicine, gene therapy, drug development, waste cleanup, climate change studies, alternative energy resources, crop improvement, structure prediction, insect resistance, improve nutritional quality, food quality research and veterinary science comparative studies.

"Support vector machines, random forests, hidden Markov models, Bayesian networks, Gaussian networks have been applied in genomics, proteomics, systems biol-

ogy, and numerous other domains" [30] and also deep learning architecture. Deep neural networks, recurrent neural networks and convolutional neural networks are used in predicting protein structure, gene expression and classifying proteins and anomalies [37].

Biological Pathways

A biological pathway is a sequence of events between molecules within a cell leading to some action that produces a change in the cell or creates a product. Some of the more important pathways in bioinformatics research are those involved in the regulation of genes, metabolism and in sending signals to and from molecules to inform or perform some task.

Pathways indicate the routes in time and transformations the molecule suffers and those it generates between its creation until its demise. Genes, proteins and other molecules in certain biological pathways may provide us information about what causes certain diseases.

Omics

Omics is a term that applies to a research field in biological sciences that ends with -omics, such as genomics, proteomics, transcriptomics, or metabolomics. It encapsulates protein structure prediction, gene expression regulation, protein classification and anomaly classification for cancer research. The "most common input data in these fields are raw biological sequences (DNA, RNA or amino acid)" and as these have variable lengths and their sequential information is important, RNN are expected to be appropriate [37].

Genomics

Genomics deals mainly with sequencing, determining the order of the pairs of nucleotide bases in a DNA chain.

DNA sequencing cannot read complete genomes at once, only about 20.000 to 30.000 pair bases at a time. We just need to know one base from each pair because the other is derived from that. In each of these shorter segments, we can do what is called *shotgun sequencing*, consisting of subdividing into smaller fragments that can be sequenced separately, and then re-assembled into a bigger segment.

When sequencing, the researchers deal with fragments that are not exclusive; they have overlapping parts. It implies multiple reads and re-reads of each base.

DNA sequencing is used in the search for genetic mutations. The variation to be discovered can be as little as a single base pair change. Genome data analysis, gene expression and gene prediction are all variants of genomics.

This area deals with assigning structures to genes, connecting diseases with genes and predicting genes from raw DNA sequence. We can have genotyping when we talk about genome sequencing and identifying genes' biological function and their part in diseases. And there is also gene expression, which has to do with correlating expression patterns, relating expression data to sequence, biological and structural data [31].

Transcriptomics

Transcriptomics is the study of all the RNA molecules within a cell and also deals with the sequencing of bases. Not the same four bases of DNA but three of them and a different one. Not all the RNA in a cell is the mRNA (messenger) that will origin the DNA. There is still a little percentage of non-coding RNA, and transcriptomics also deals with these. These other types of RNA can be transfer RNA (tRNA) or ribosomal RNA (rRNA). All three of these are present in all prokaryote and eukaryote organisms. Nevertheless, there are also small quantities of other types of RNA.

Proteomics

Proteomics is the analysis of proteins and proteomes (all proteins produced by an organism) and, more specifically, protein purification and mass spectrometry. One of the main applications is discovering new drugs to treat diseases, finding coupling possibilities in the protein structure that disables or inactivates enzymes or malign parts of molecules.

It deals with the comparison of sequences, sequence prediction and predictions inferred from sequences. Roles of proteins in biological systems, 2D and 3D structures, geometry measurements and intermolecular interactions are all possibilities. [31]

Table 2.1: IUPAC's one- and three-letter amino acid codes

Amino Acid	Code	Code
<i>alanine</i>	ala	A
<i>arginine</i>	arg	R
<i>asparagine</i>	asn	N
<i>aspartic acid</i>	asp	D
<i>asparagine or aspartic acid</i>	asx	B
<i>cysteine</i>	cys	C
<i>glutamic acid</i>	glu	E
<i>glutamine</i>	gln	Q
<i>glutamine or glutamic acid</i>	glx	Z
<i>glycine</i>	gly	G
<i>histidine</i>	his	H
<i>isoleucine</i>	ile	I
<i>leucine</i>	leu	L
<i>lysine</i>	lys	K
<i>methionine</i>	met	M
<i>phenylalanine</i>	phe	F
<i>proline</i>	pro	P
<i>serine</i>	ser	S
<i>threonine</i>	thr	T
<i>tryptophan</i>	trp	W
<i>tyrosine</i>	tyr	Y
<i>valine</i>	val	V

Metabolomics

Metabolomics give a functional reading of the organisms' physiological state, from the study of metabolites, which are the final or transitional products of metabolism. We can see an infographic about everything metabolomics in Figure 2.2.

It has applications in drug toxicity studies, disease modelling and most importantly fingerprinting and pathway discovery.

Metabolomics provides a better understanding of how cells work, as well as identifying new or important alterations in particular metabolites. We can obtain new hypotheses and new targets for biotechnology through the analysis and data mining of metabolomic data sets [49].

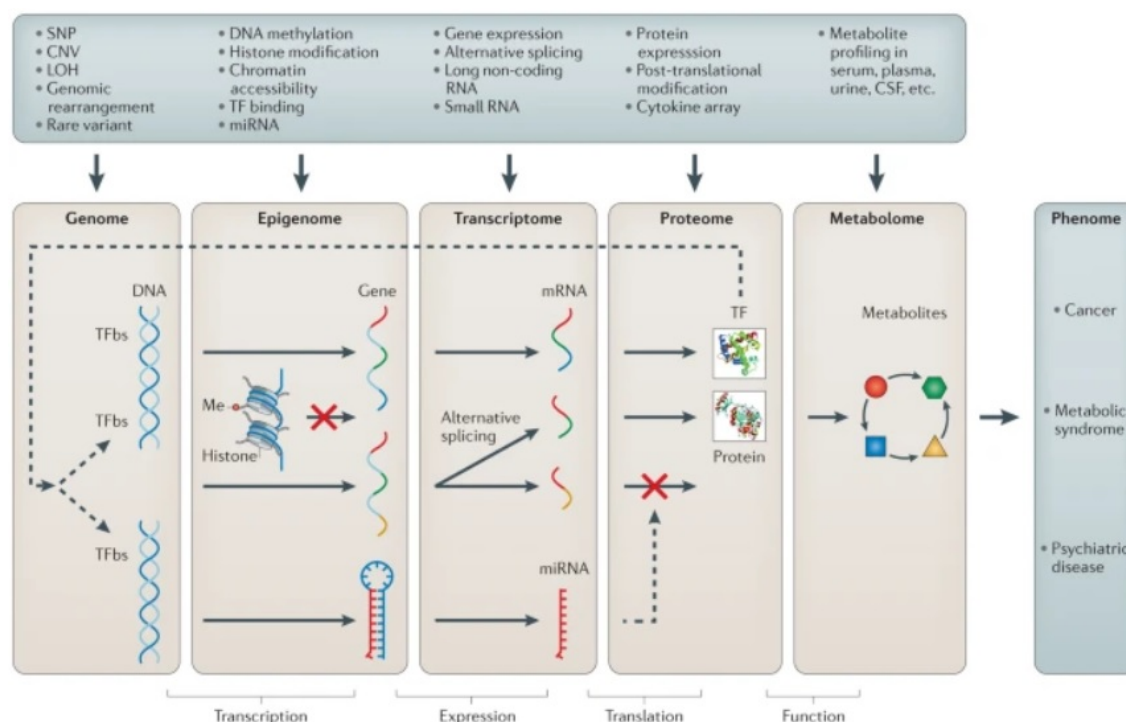


Figure 2.2: Omics from the genome to the phenome. Source: [48]

Sequence representations

The nucleotide and amino acid sequences are represented by their IUPAC one-letter codes (Table 2.1). Margaret O. Dayhoff was the first person to have ever used single letters to represent amino acids. Until then, and even after, the standard three letters were used.

If we choose to represent a sequence in plain format, it will be like this¹:

```
ACAAGATGCCATTGTCCCCGGCCTCCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCC
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGC
CTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCAGGCCAGTGCCGGGCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGACCCCCCAGCAATCCGCGCGCCGGGACAGAATGCC
CTGCAGGAACCTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

Cheminformatics

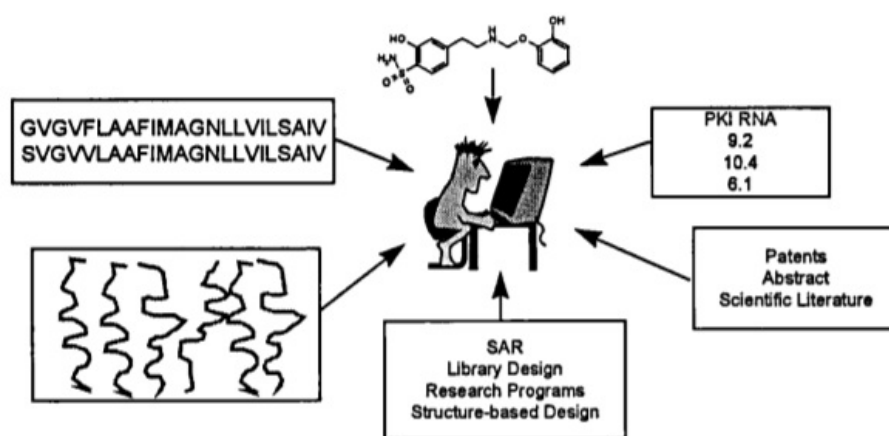


Figure 2.3: Cheminformatics viewed by Frank K. Brown. Source: [7]

The term “cheminformatics” was first coined by Dr Frank K. Brown in 1998. Figure 2.3 depicts his approach. In his article for the Annual Reports of Medicinal Chemistry, he defined Cheminformatics as

“The mixing of information resources to transform data into information, and information into knowledge, for the intended purpose of making better decisions faster in the arena of drug lead identification and optimisation.” [7]

One year later, Greg Paris related it to everything it has to do with:

“Chem(o)informatics is a generic term that encompasses the design, creation, organisation, management, retrieval, analysis, dissemination, visualisation and use of chemical information.” [58]

The tools of cheminformatics are used to describe chemical structures for use in computer databases and to analyze the connections between structure and molecular properties [59].

¹https://www.genomatix.de/online_help/help/sequence_formats.html

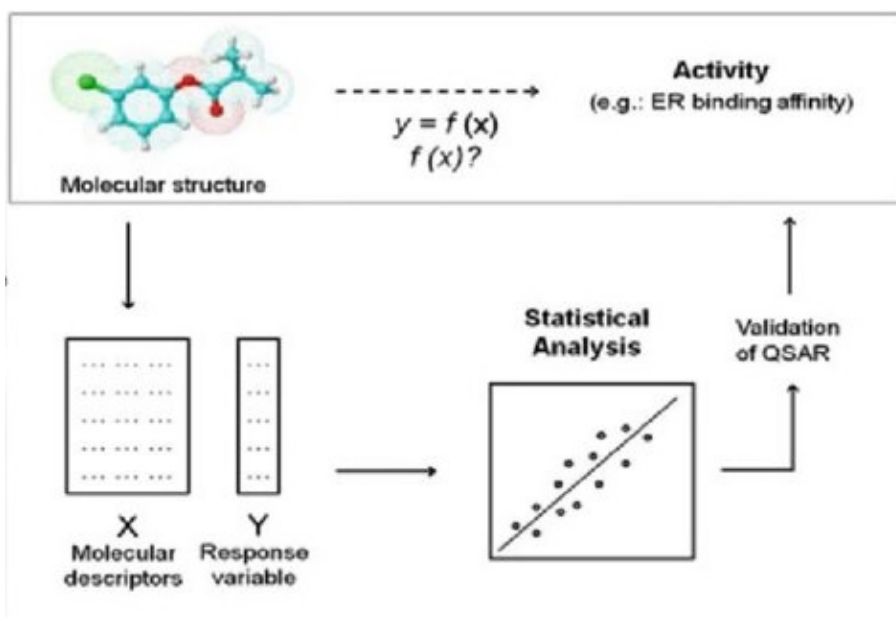


Figure 2.4: QSAR / QSPR flow diagram. Source: adapted from [17]

In the beginning, cheminformatics used local models based on linear, and later multilinear regression that only used a few known data features and was only applicable to a close-knit group of few related compounds. They were used mainly for quantitative structure–activity relationships (QSAR) or quantitative structure–property relationships (QSPR) [38]. Figure 2.4 shows a QSAR/QSPR flow diagram.

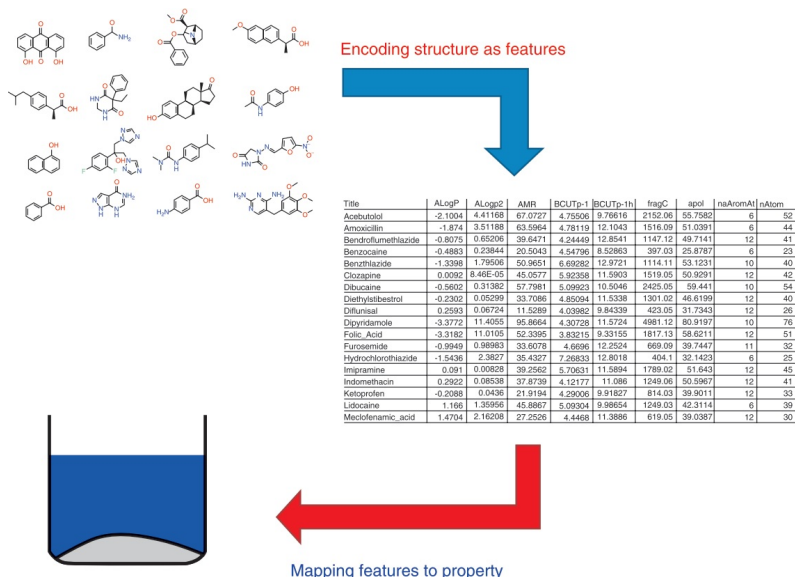


Figure 2.5: Cheminformatics basic steps. Source: [38]

In chemistry, both the structure and the properties of matter are the sum of what makes it (atoms, groups of atoms and molecules). The way the atoms or other com-

ponents of matter relate to each other, proximity, and more, determines the angles between molecules, their geometry and, consequently, their properties. Different properties of the matter influence its activity. Refer to Figure 2.5 to understand Cheminformatics basic steps.

Molecule representations

To study and derive conclusions from known chemical compounds, the first thing to do is to represent the molecules. We can represent them one-dimensionally, in 2D or 3D.

In 1D, it is just a text string with the components' chemical symbols, following a set of rules. Rules of where to place the radicals, connected atoms, parenthesis and how to represent the different types of bonds and chiral carbon atoms. Some standards for these representations exist like SMILES, InChi (Figure 2.6).

The IUPAC International Chemical Identifier (InChI) is a chemical substance identifier in text format, developed by NIST (National Institute of Standards and Technology) and IUPAC (International Union of Pure and Applied Chemistry).

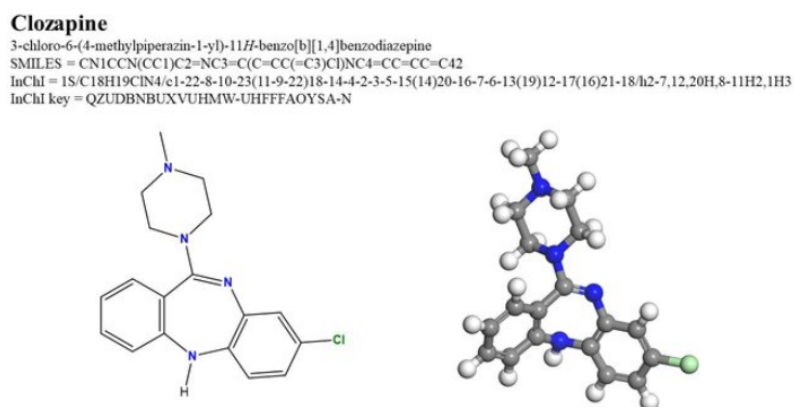


Figure 2.6: Example of chemical structure representations. Source: [9]

At first glance, not much stereochemistry can be inferred from a one-dimensional representation. 1D is also used to store descriptors of physicochemical properties like solubility, melting and boiling points, partition coefficients, and individual properties such as molecular weight, number of donor, and acceptor electrons, and number of aromatic rings [33].

A two-dimensional representation would be like a graph, with the atoms as nodes and the bonds as edges. Although we could add more info into the weights of the edges, like bond length or order, angle and other properties, it could still not be enough to determine the spatial coordinates of the atoms.

Some applications have developed different forms of representing the molecules three-dimensionally. The representations are usually with 'stick' or 'ball-and-stick' models. In some cases approximate models of electron density distributions [33]. The 3D comparisons between compounds can find similarities that were not present in the 2D representation, but they are very computationally demanding, and in most cases, 2D is a suitable compromise.

SMILES

SMILES is a unidimensional ASCII string chemical representation specification and file format (.smi). It exists since the 1980s and is more human-readable than IUPAC's InChI. Its final representation depends on what atom to put as node root and at each branch point which path to follow first.

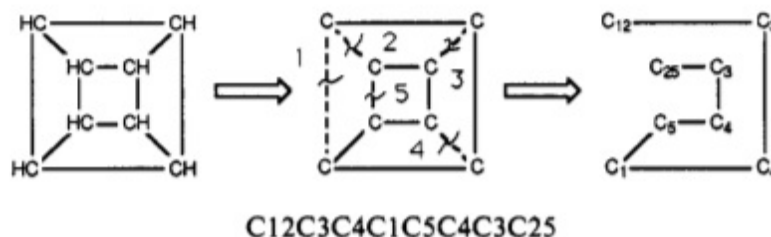


Figure 2.7: Cangen algorithm for SMILES representation of Cubane (C₈H₈). Source: adapted from [61]

In 1989, an algorithm was proposed to create canonised SMILES representations of molecules easily [61]. This algorithm (Figure 2.7), begins by breaking each cycle by removing a bond. Numerical indices then mark broken bonds, and the atoms of broken bonds are referenced by a concatenation of the indices of all the bonds they make with the other atoms. Then we get a linearised version of the molecule, and a string is produced by adding the chemical symbol and bond indices of the atom at one end and appending the rest of the atoms until the end of the chain in the same manner. We can say it is obtained through a depth-first traversal.

InChi

InChI is IUPAC's representation identifier and overcomes various SMILES problems like its semi-proprietary and incompleteness. The atoms are numbered according to the molecular formula. The connectivity information is similar to SMILES but adds stereochemistry and describes molecules in five layers:

- Main layer or molecular formula
- Skeletal layer or topology (atom connectivity)
- Hydrogen layer (bond orders and hydrogen locations)
- Charge Layer
- Stereochemistry Layer

In the first layer, the chemical formula is represented beginning with carbon atoms, then hydrogens, then all other elements in alphabetical order, according to IUPAC conventions. The second layer is prefixed with /c and describes the connections between the skeletal atoms, listing the canonical numbers of the atoms in the chain and branches between parentheses. The hydrogen layer is prefixed with /h and enumerates the bonds between the atoms. The charge layer describes the elements' net charge. The last layer contains sublayers representing double bond stereochemistry and tetrahedral stereochemistry. Some of these layers include other sublayers that contain prefixes [21].

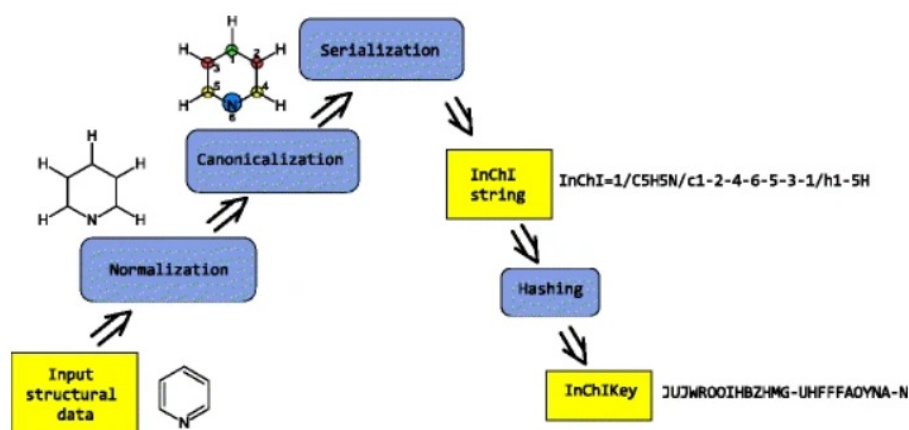


Figure 2.8: InChIKey generation workflow diagram.

Source: [21]

InChIKey

InChIKey is a compact chemical identifier derived from InChI. It is an encoded version of the hash codes calculated from a source InChI string and is always 27-symbols long. Therefore, it is a far more useful identifier to be used in indexing databases and internet searches.

It is composed of three blocks separated by a dash. The first block is 14-characters long and encodes the core molecular constitution. The second block is 10-characters long and has two parts. The first six letters encode the advanced structural features (stereochemistry, isotope, ligations) and the last two letters represent InChI's standard and version. A final block is composed of one letter that maps the net charge or (de)protonation of the molecule [21].

In Figure 2.8, we see the general workflow of InChI creation and InChIKey generation for pyridine.

SDF or MDL molfile

SDF is the acronym of Structural Data Files which is a file format to represent chemical compounds according to the MDL Molfile format, which is an ontology for chemical compounds, on how to represent the atoms, bonds, connectivity and coordinates. SDF can include associated data.

All molfiles have a header and a connection table. The connection table is divided into two blocks, the Atom Block and the Bond Block.

The header block has two lines. The first line is the chemical formula and the second line has the program that made the file, the date (MMddyyhhmm) and if we have 2D or 3D coordinates.

After that comes the count block that begins with the number of atoms and bonds and ends with the version number of the molfile.

The atom block is of variable length and has one line for each atom in the molecule, beginning with the coordinates and followed by the chemical symbol of the atom.

The bond block is also of variable length and has one line for each bond in the molecule. The first two values tell us between which atoms is the bond and the third value indicates the type of bond.

The molecule ends with "M END" and we can have a properties block after that and before the end of the file marked with "\$\$\$\$"

We present an example for aspirin:

```

C9H8O4
APtclcactv02092107253D 0 0.00000 0.00000

21 21 0 0 0 0 0 0 0 0999 V2000
 2.2393 -0.3791 0.2630 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.8424 1.9231 -0.4249 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2.8709 0.8456 0.2722 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2.1751 1.9935 -0.0703 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-3.4838 0.4953 -0.0896 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.8910 -0.4647 -0.0939 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.1908 0.6991 -0.4402 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-0.9633 -1.8425 -0.4185 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1.6531 0.8889 1.3406 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.8857 -2.8883 0.2267 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.2090 -1.7720 -0.1069 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-2.0185 0.6853 0.2071 C 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-1.1189 0.6285 -0.7886 O 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.3962 -3.7219 0.2035 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2.7867 -1.2719 0.5268 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0.3069 2.8224 -0.6911 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 3.9130 0.9108 0.5482 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2.6781 2.9492 -0.0604 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-3.7360 -0.5623 -0.0120 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-4.0763 1.0637 0.6273 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-3.6988 0.8471 -1.0986 H 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 6 7 2 0 0 0 0
 1 6 1 0 0 0 0
 6 11 1 0 0 0 0
 2 7 1 0 0 0 0
 7 13 1 0 0 0 0
 1 3 2 0 0 0 0
10 11 1 0 0 0 0
 8 11 2 0 0 0 0
 2 4 2 0 0 0 0
12 13 1 0 0 0 0
 5 12 1 0 0 0 0
 9 12 2 0 0 0 0
 3 4 1 0 0 0 0
 1 15 1 0 0 0 0
 2 16 1 0 0 0 0
 3 17 1 0 0 0 0
10 14 1 0 0 0 0
 4 18 1 0 0 0 0
 5 19 1 0 0 0 0
 5 20 1 0 0 0 0
 5 21 1 0 0 0 0

M END
$$$$

```

We can see that the file was created on February 2nd 2021 at 07h25 EST and corresponds to 3D coordinates. The count block tells us it has 21 atoms and 21 bonds. The atom block has 21 lines and the bond block also.

Applications in cheminformatics

From the stored molecule representations, we extract features, also known as descriptors. Everything can be a descriptor, from the number of total atoms to a certain radical's geometry attached to the molecule.

Data mining models will map the features to molecule properties using trained models for classification and regression. Properties like dipole moments, polarizabilities or vibrational frequencies, are better obtained using quantum chemistry theory and calculations than with data mining. Data mining helps understand the properties within a broader system, where physical chemistry methods are not enough [38].

Methods ^a	Examples of application(s)
Quantitative, qualitative and graphical representation of chemical compounds (calculation of descriptors)	<ul style="list-style-type: none"> • Communication of chemical data. • Development of quantitative and qualitative structure-property (activity) relationships.
Collection and storage of chemical data	<ul style="list-style-type: none"> • Organization of chemical databases. • Organization and analysis of high-throughput screening results.
Structure search	<ul style="list-style-type: none"> • Data mining. • Virtual screening.
Analysis of molecular similarity and diversity: diversity analysis	<ul style="list-style-type: none"> • Comparison of chemical libraries. • Design and optimization of compound databases. • Acquisition of compound data sets. • Similarity searching and virtual screening
Calculation and prediction of compound properties	<ul style="list-style-type: none"> • Optimization of hit and lead molecules. • Assessment of lead- and drug-likeness. • Modeling of ADME-Tox properties. • Prediction of the metabolism. • Filtering compounds.
Prediction of biological activity and <i>in vivo</i> compound characteristics	<ul style="list-style-type: none"> • Identification of hit and lead molecules. • Virtual screening: filtering compound and data set (including <i>data shaving</i>).
Computer-assisted synthesis design	<ul style="list-style-type: none"> • Ligand- and structure-based design: Assessment of synthetic feasibility. • <i>De novo</i> design.
Visualization of chemical information	<ul style="list-style-type: none"> • Communication and pattern recognition. • Assessment of chemical diversity. • Compound classification and selection. • Interpretation of high-throughput data (virtual and experimental).
Molecular modeling	<ul style="list-style-type: none"> • Docking: including automated data analysis of protein-ligand and protein-protein interactions. • Structure-based design (e.g., fragment-based and <i>de novo</i>). • (Accurate) calculation of binding affinities. • Prediction of the three-dimensional structure of molecular targets.

^a Several methods have more than one application.

Table 2.2: Cheminformatics methods. Source: [33]

Many areas of application exist in Cheminformatics. In Table 2.2, we can see some of the methods. In the following paragraphs, we will talk about some of the areas of application in cheminformatics of a few data mining and machine learning techniques, such as support vector machines, random forests, K-nearest neighbours, artificial neural networks, and the naïve Bayes classifier.

K-nearest neighbours depends heavily on the k parameter and are sensitive to local structures, being ideal for predicting properties with a strong locality, such as protein functions [38].

Bioactivity prediction

Support Vector Machines and K-nearest neighbours have been used to predict numerous molecules' bioactivity when adding new elements to it, like finding agonists for receptors or inhibitors for enzymes. Artificial Neural Networks can also be used but they are vulnerable to overfitting and learning noise and may not be suitable for new data [38].

Toxicological prediction

The naïve Bayes classifier is very good in predicting biological properties like toxicity. Support Vector Machines are among the most popular machine-learning methods in cheminformatics and can also predict various toxicity-related properties [38].

The properties considered in this area are known as ADMET (absorption, distribution, metabolism, excretion and toxicity). Although there are good amounts of reliable experimental data in databases concerning metabolism and toxicity, for the other properties such is not the case, making it difficult to train models [23].

Pharmacological prediction

The naïve Bayes classifier is successfully used for protein prediction and classification for drug-like molecules. K-nearest neighbours is used in finding anti-inflammatories and anti-cancer drugs [38]. The properties mentioned in bioactivity (section 2) can also be used in drug research as can be discoveries for athletic performance enhancement in which many methods are used.

Physicochemical properties

Support Vector Machines successfully predict properties like solubility, acid-base, partition and density coefficients and melting point. K-nearest neighbours is also very successful in predicting those properties and also of the boiling point. Solubility and solvent discovery can also be predicted with the use of random forests [38]. According to [23], there is plenty of information in databases and literature regarding these properties and methods.

2.1 Repositories and their API

To search for information regarding the problems at hand, we will fetch data available on free databases on the web. Some of these databases have some common information; others are aggregators of others and proxies to others. Despite this, there is information only available on some and not all repositories, so we have to visit several databases to gather the most information possible. The access to these repositories is made through web API made available by them. An API has formatted and specific requests made possible to fetch and upload data. In our case, only the fetching will be necessary.

We start with some repositories from NCBI with the Entrez Programming Utilities (E-utilities). Entrez is a molecular biology database system providing integrated access to nucleotide and protein sequence data, gene-centred and genomic mapping information and 3D structure data. It is maintained by the National Center for

Biotechnology Information (NCBI) and covers over 20 databases, including chemical compounds from PubChem and nucleotide sequence data from GenBank.

Another group of repositories come from the EMBL-EBI, the European Bioinformatics Institute, part of the European Molecular Biology Laboratory. These repositories are the genome database Ensembl, the comprehensive resource for protein sequence UniProt and the open data resource of binding, functional and ADMET bioactivity data ChEMBL.

GenBank [2] [3] [50]

GenBank is the US National Institutes of Health genetic sequence database and one of the most used globally. It contains all the publicly available DNA sequences, with annotations. GenBank is the NCBI (National Center for Biotechnology Information) part of the International Nucleotide Sequence Database Collaboration, together with the DNA DataBank of Japan, and the European Nucleotide Archive. These three organisations exchange data daily.

A GenBank release occurs every two months. The current genetic sequence data bank was released on August 18th, 2021 as Release 245.0. It comprises more than 230 million DNA sequences with more than 940 thousand million base pairs.

We present below an example of an API request of a summary of a Genbank record:
GET <https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=gene&id=22>

and its reply (fragment):

```
{<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE eSummaryResult PUBLIC "-//NLM//DTD esummary gene 20150202//EN" "https://
eutils.ncbi.nlm.nih.gov/eutils/dtd/20150202/esummary_gene.dtd">
<eSummaryResult>
  <DocumentSummarySet status="OK">
    <DbBuild>Build210210-2100m.1</DbBuild>
    <DocumentSummary uid="22">
      <Name>ABCB7</Name>
      <Description>ATP binding cassette subfamily B member 7</Description>
      <Status>0</Status>
      <CurrentID>0</CurrentID>
      <Chromosome>X</Chromosome>
      <GeneticSource>genomic</GeneticSource>
      <MapLocation>Xq13.3</MapLocation>
      <OtherAliases>ABC7, ASAT, Atm1p, EST140535</OtherAliases>
      <OtherDesignations>ATP-binding cassette sub-family B member 7,
        mitochondrial|ABC transporter 7 protein|ATP-binding cassette
        transporter 7|ATP-binding cassette, sub-family B (MDR/TAP), member 7</
        OtherDesignations>
      <NomenclatureSymbol>ABCB7</NomenclatureSymbol>
      <NomenclatureName>ATP binding cassette subfamily B member 7</
        NomenclatureName>
      <NomenclatureStatus>Official</NomenclatureStatus>
      <Mim>
        <int>300135</int>
      </Mim>
    </DocumentSummary>
  </DocumentSummarySet>
</eSummaryResult>
```

```

<GenomicInfo>
  <GenomicInfoType>
    <ChrLoc>X</ChrLoc>
    <ChrAccVer>NC_000023.11</ChrAccVer>
    <ChrStart>75156282</ChrStart>
    <ChrStop>75051047</ChrStop>
    <ExonCount>16</ExonCount>
  </GenomicInfoType>
</GenomicInfo>
<GeneWeight>3493</GeneWeight>
<Summary>The membrane-associated protein encoded by this gene is a member
of the superfamily of ATP-binding cassette (ABC) transporters. ABC
proteins transport various molecules across extra- and intra-cellular
membranes. ABC genes are divided into seven distinct subfamilies (ABC1,
MDR/TAP, MRP, ALD, OABP, GCN20, White). This protein is a member of
the MDR/TAP subfamily. Members of the MDR/TAP subfamily are involved in
multidrug resistance as well as antigen presentation. This gene
encodes a half-transporter involved in the transport of heme from the
mitochondria to the cytosol. With iron/sulfur cluster precursors as its
substrates, this protein may play a role in metal homeostasis.
Mutations in this gene have been associated with mitochondrial iron
accumulation and isodicentric (X)(q13) and sideroblastic anemia.
Alternatively spliced transcript variants encoding multiple isoforms
have been observed for this gene. [provided by RefSeq, Nov 2012]</
Summary>
<ChrSort>X</ChrSort>
<ChrStart>75051047</ChrStart>
<Organism>
  <ScientificName>Homo sapiens</ScientificName>
  <CommonName>human</CommonName>
  <TaxID>9606</TaxID>
</Organism>
<LocationHist>
  <LocationHistType>
    <AnnotationRelease>109.20201120</AnnotationRelease>
    <AssemblyAccVer>GCF_000001405.39</AssemblyAccVer>
    <ChrAccVer>NC_000023.11</ChrAccVer>
    <ChrStart>75156282</ChrStart>
    <ChrStop>75051047</ChrStop>
  </LocationHistType>
  ...
  <LocationHistType>
    <AnnotationRelease>105</AnnotationRelease>
    <AssemblyAccVer>GCF_000306695.2</AssemblyAccVer>
    <ChrAccVer>NC_018934.2</ChrAccVer>
    <ChrStart>74269045</ChrStart>
    <ChrStop>74165915</ChrStop>
  </LocationHistType>
</LocationHist>
</DocumentSummary>
</DocumentSummarySet>
</eSummaryResult>

```

If we want the full record we should use the efetch utility:

```
GET https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=
gene&id=22
```

We can define the return mode:

```
GET https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=
gene&id=22&retmode=xml
```

We can also search for terms with the esearch utility:

```
GET https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db
=gene&term=corona
```

A GenBank file may comprise several sequences and each sequence has an accession number. This is a unique identifier for different versions of the sequence and not for each sequence. All repositories with sequence information use accession numbers. The accession number has a prefix that identifies the molecule type (see Table 2.3 for RefSeq accession prefixes).

Table 2.3: NCBI GenBank RefSeq accession number prefixes and molecule types.

Prefix	Type	Comment
AC_	Genomic	Complete genomic molecule, usually alternate assembly
NC_	Genomic	Complete genomic molecule, usually reference assembly
NG_	Genomic	Incomplete genomic region
NT_	Genomic	Contig or scaffold, clone-based or WGS ^a
NW_	Genomic	Contig or scaffold, primarily WGS ^a
NZ_ ^b	Genomic	Complete genomes and unfinished WGS ^a data
NM_	mRNA	Protein-coding transcripts (usually curated)
NR_	RNA	Non-protein-coding transcripts
XM_ ^c	mRNA	Predicted model protein-coding transcript
XR_ ^c	RNA	Predicted model non-protein-coding transcript
AP_	Protein	Annotated on AC_ alternate assembly
NP_	Protein	Associated with an NM_ or NC_ accession
YP_ ^c	Protein	Annotated on genomic molecules without an instantiated transcript record
XP_ ^c	Protein	Predicted model, associated with an XM_ accession
WP_	Protein	Non-redundant across multiple strains and species

^aWhole Genome Shotgun sequence data

^bAn ordered collection of WGS sequence for a genome

^cComputed

A GenBank record id has some prefixes that we need to understand. Table 2.3 clarifies what they mean.

PubChem [28]

PubChem is a public chemistry database started in 2004 by the National Institutes of Health (NIH). It is updated constantly.

The repository stores around 110 million compounds and 270 million substances, mostly small molecules. It also comprises chemically-modified macromolecules, nucleotides, peptides, lipids, and carbohydrates. Their principal identifiers are *cid* for compounds and *sid* for substances.

We present below an example of an API request asking for Aspirin properties:

```
GET https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/2244/
property/MolecularFormula,MolecularWeight,CanonicalSMILES,
IsomericSMILES,InChI,InChIKey,IUPACName,XLogP,ExactMass,MonoisotopicMass,
TPSA,Complexity,Charge,HBondDonorCount,HBondAcceptorCount,
RotatableBondCount,HeavyAtomCount,IsotopeAtomCount,AtomStereoCount,
DefinedAtomStereoCount,UndefinedAtomStereoCount,BondStereoCount,
DefinedBondStereoCount,UndefinedBondStereoCount,CovalentUnitCount,
Volume3D,XStericQuadrupole3D,YStericQuadrupole3D,
ZStericQuadrupole3D,FeatureCount3D,FeatureAcceptorCount3D,
FeatureDonorCount3D,FeatureAnionCount3D,FeatureCationCount3D,
FeatureRingCount3D,FeatureHydrophobeCount3D,ConformerModelRMSD3D,
EffectiveRotorCount3D,ConformerCount3D,Fingerprint2D/json
```

and its reply:

```
{
  "PropertyTable": {
    "Properties": [
      {
        "CID": 2244,
        "MolecularFormula": "C9H8O4",
        "MolecularWeight": 180.16,
        "CanonicalSMILES": "CC(=O)OC1=CC=CC=C1C(=O)O",
        "IsomericSMILES": "CC(=O)OC1=CC=CC=C1C(=O)O",
        "InChI": "InChI=1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12)",
        "InChIKey": "BSYNRYMUTXBXSQ-UHFFFAOYSA-N",
        "IUPACName": "2-acetyloxybenzoic acid",
        "XLogP": 1.2,
        "ExactMass": 180.042259,
        "MonoisotopicMass": 180.042259,
        "TPSA": 63.6,
        "Complexity": 212,
        "Charge": 0,
        "HBondDonorCount": 1,
        "HBondAcceptorCount": 4,
        "RotatableBondCount": 3,
        "HeavyAtomCount": 13,
        "IsotopeAtomCount": 0,
        "AtomStereoCount": 0,
        "DefinedAtomStereoCount": 0,
        "UndefinedAtomStereoCount": 0,
        "BondStereoCount": 0,

```

```

    "DefinedBondStereoCount": 0,
    "UndefinedBondStereoCount": 0,
    "CovalentUnitCount": 1,
    "Volume3D": 136,
    "XStericQuadrupole3D": 3.86,
    "YStericQuadrupole3D": 2.45,
    "ZStericQuadrupole3D": 0.89,
    "FeatureCount3D": 5,
    "FeatureAcceptorCount3D": 3,
    "FeatureDonorCount3D": 0,
    "FeatureAnionCount3D": 1,
    "FeatureCationCount3D": 0,
    "FeatureRingCount3D": 1,
    "FeatureHydrophobeCount3D": 0,
    "ConformerModelRMSD3D": 0.6,
    "EffectiveRotorCount3D": 3,
    "ConformerCount3D": 10,
    "Fingerprint2D": "
      AAADccBwOAAAAAAAAAAAAAAAAAAAAAAAAAAAwAAAAAAAAABAAAAGgAACAAADASAmAAYDoAABgC
      IAiDSCAACCAAKIAAiAEGCMgMJzaENRqCe2Cl4BEIuYeIyCCOAAAAAAAAIAAAAAAAAAABAAAAAAAAAA
      ==
    "
  }
]
}
}

```

e!Ensembl [42][65]

Ensembl, from EBI-EMBL, is a genome browser for vertebrate genomes that supports research in comparative genomics, evolution, sequence variation and transcriptional regulation. It annotates genes, computes multiple alignments, predicts regulatory function and collects disease data.

The current Release 104, from May 2021, has data from 318 animal species. Release 106 is planned to arrive before the end of 2021.

We present below an example of an API request:

GET <http://rest.ensembl.org/sequence/id/ENSP00000288602?content-type=application/json>
and its reply:

```

{
  "query": "ENSP00000288602",
  "seq": "
    MAALSGGGGGGAEPGQALFNGDMEPEAGAGAAAASSAADPAIPEEVWNIKQMIKLTQEHEIALLDKFGGEHNPPSIYL
    EAYEYTSKLDALQREQQLLES LGNGTDFSVSSASMDT VTS SSSSSLSVLPSSLV FQNPTDVARSNPKSPQKPIVRVFLP
    NKQRTVVPARCGVTVRDSLKALMMRGLIPECCAVYRIQDGEKKPIGWDTDISWLTGEELHVEVLENVPLTTHNFVRKTFFTL
    AFCDFCRKLLFQGFRCQTCGYKFHQRCSTEVPLMVCVNYDQLDLLFVSKFFEHPPIQEEASLAETALTS GSSPSAPASDSIGP
    QILTSPSPSKSIPIQPFRPADEHNRNQFGQRDRSSAPNVHINTIEPVNIDDLIRDQGFGRGDGAPLNLQMRCLRKYQSRTPS
    PLLHVPSEIVDFEPGPVFRGSTTGLSATPPASLPGSLTNVKALQKSPGPQREKSSSSSEDRNRMKTLGRRDSSDDWEIPD
    GQITVQQRIGSGSGFTVYKKGWHDVAVKMLNVTAPTQQQLQAFKNEVGVLKTRHVNILLFMGYSTKPQLAIVTQWCEGSSL
    YHHLHIETKFEMIKLIDIAARQTAQGM DYLHAKSIIHRDLKSNINIFLHEDLTVKIGDFGLATVKSRWSGSHQFEQLSGSILWM
    APEVIRMQDKNPYSFQSDVYAFGIVLYELMTGQLPYSNINNRDQIIFMVGRGYLSPDL SKVRSNCPKAMKRLMAECLKKRDE
    RPLFPQILASIELLARSLPKIHRASEPSLNRAGFQTEDFSLYACASPKTPIQAGGYGAFPVH",

```

```

"molecule": "protein",
"desc": null,
"version": 7,
"id": "ENSP00000288602"
}

```

KEGG [24][25][26]

KEGG is a molecular-level database with large datasets produced, especially, by genome sequencing. It means Kyoto Encyclopedia of Genes and Genomes and is one of the largest banks of biological pathways, currently holding more than 765 thousand.

The latest release is 99.1 from August 1st, 2021. It currently stores data from almost 7000 organisms (545 eukaryotes, 6095 bacteria, 338 archaea), 349 viruses and almost 35 thousand genes. It also comprises small molecule and glycan information, more than 11 thousand chemical and biochemical reactions, complemented with human disease and drug information.

We present below an example of an API request for a human gene:

GET <http://rest.kegg.jp/get/hsa:10458>

and its reply:

```

{
ENTRY      10458          CDS      T01001
NAME       BAIAP2, BAP2, FLAF3, IRSP53, WAML
DEFINITION (RefSeq) BAR/IMD domain containing adaptor protein 2
ORTHOLOGY K05627 BAI1-associated protein 2
ORGANISM   hsa Homo sapiens (human)
PATHWAY    hsa04520 Adherens junction
           hsa04810 Regulation of actin cytoskeleton
           hsa05130 Pathogenic Escherichia coli infection
           hsa05135 Yersinia infection
NETWORK    nt06135 Cytoskeletal regulation (viruses and bacteria)
ELEMENT    N01094 Escherichia Eae/Tir/TccP to Actin signaling pathway
BRITE      KEGG Orthology (KO) [BR:hsa00001]
           09140 Cellular Processes
           09144 Cellular community - eukaryotes
           04520 Adherens junction
           10458 (BAIAP2)
           ...
           Membrane trafficking [BR:hsa04131]
           Endocytosis
           Bin/Amphiphysin/Rvs (BAR) family proteins
           I-BAR proteins

```

	10458 (BAIAP2)
POSITION	17q25.3
MOTIF	Pfam: IMD SH3_2 SH3_9 SH3_1 BAR RasGAP Peptidase_Mx1 BAR_3
DBLINKS	NCBI-GeneID: 10458 NCBI-ProteinID: NP_059345 OMIM: 605475 HGNC: 947 Ensembl: ENSG00000175866 Vega: OTTHUMG00000177698 Pharos: Q9UQB8(Tbio) UniProt: Q9UQB8
STRUCTURE	PDB: 3RNJ 4JS0 2YKT 1Y20 1WDZ 6BQT 6BD2
AASEQ	552 MSLSRSEEMHRLTENVYKTIMEQFNPSLRNFIAMGKNYEKALAGVTYAAKGYFDALVKMG ELASESQSGKELGDVLFQMAEVHRQIQNQLEEMLKSFHNELLTQLEQKVELDSRYLSAAL KKYQTEQRSGDALDKCQAEKLRKKSQGSKNPQKYSKELQYIDAISNKQGELENYVS DGYKTALTEERRRFCFLVEKQCAVAKNSAAYHSGKELLAQKLPWQQACADPSKIPERA VQLMQQVASNGATLPSALSASKSNLVISDPIPGAKPLPVPELAPFVGRMSAQESTPIMN GVTGPDGEDYSPWADRKAAQPKSLSPQSQSKLSDSYSNTLPVRKSVTPKNSYATTENKT LPRSSMAAGLERNGMRVKAIFSHAAGDNSTLLSFKEGDLITLLVPEARDEWHYGESEK TKMRGWFPFSYTRVLSDSDGSDRLHMSLQQGKSSSTGNLLDKDDLAIPPPDYGAASRAFFA QTASGFKQRPYSVAVPAFSQGLDDYGARSMSRNPFAHVQLKPTVTNDRCDLSAQGPEGRE HGDGSARTLAGR
NTSEQ	1659 atgtctctgtctcgctcagaggagatgcaccggctcacggaaaatgtctataagaccatc atggagcagttcaaccctagcctccggaactcatcgccatggggaagaattacgagaag gactggcaggtgtgacgtatgcagccaaaggctactttgacgccctgggaagatgggg gagctggccagcgagagccagggctccaaagaactcggagacgttctctccagatggct gaagtccacaggcagatccagaatcagctggaagaaatgctgaagtcttttcacaacgag ctgcttacgcagctggagcagaaggtggagctggactccaggtatctgagtgctgctg aagaaataccagactgagcaaggagcaaaggcgacgccctggacaagtgtcaggctgag ctgaagaagcttcggaagaagagccagggcagcaagaatcctcagaagtactcggacaag gagctgcagtacatcgacgccatcagcaacaagcagggcgagctggagaattacgtgtcc gacggctacaagaccgactgacagaggagcgcaggcgcttctgcttctctggaggagaag cagtgccgctggccaagaactccgcgccctaccactccaagggaaggagctgctggcg cagaagctgccgctgtggcaacaggcctgtgccgaccccagcaagatcccggagcgcg gtgcagctcatgcagcaggtggcagcaacggcgccaccctcccagcgcctgtcggcc tccaagtccaactggtcatttccgacccattccggggccaagcccctgcccgtgccc cccagctggcaccgttctgtggggcgatgtctgccaggagagcacaccatcatgaac ggcgtcacaggcccgatggcgaggactacagcccgtgggctgaccgcaaggctgcccag cccaatcccctgtctcctccgagctctcagagcaagctcagcgactcctactccaacaca ctcccgtgcaagagcgtgacccccaaaaaacagctatgccaccaccgagaacaagact ctgcctcgtcagctccatggcagccggcctggagcgcaatggccgtatgcccgtggaag gccatcttctcccacgtgctggggacaacagcaccctcctgagcttcaaggagggtgac ctattaccctgctggtgctgaggcccgcgatggctggcactacggagagagtgagaag accaagatgcccggctggtttcccttctctacaccgggtcttgacagcgatggcagct

```

gacaggctgcacatgagcctgcagcaaggaagagcagcagcacgggcaacctcctggac
aaggacgacctggccatcccacccccgattacggcgccgcctcccgggccttccccgcc
cagacggccagcggcttcaagcagaggccctacagtgtggcctgcccgccttctcccag
ggcctggatgactatggagcgcggtccatgagcaggaatccctttgccacgtccagctg
aagccgacagtgaccaacgacaggtgtgatctgtccgccaagggccagaaggccgggag
cacggggatgggagcggccgcaccctggctggaagatga
///

```

RCSB protein data bank [4]

The Protein Data Bank (PDB) was the first free digital database with medical and biological data, hence the acronym for Research Collaboratory for Structural Bioinformatics. It was established in 1971 at Brookhaven National Laboratory and originally contained seven structures. It currently holds 181780 Biological Macromolecular Structures.

We can explicitly request information directly from a particular database, like PubMed, UniProt and DrugBank by their specific identifier or to PDB:

```
GET https://data.rcsb.org/rest/v1/core/pubmed/1RH7
```

getting the following reply (fragment):

```

{
  "rcsb_id": "15155948",
  "rcsb_pubmed_container_identifiers": {
    "pubmed_id": 15155948
  },
  "rcsb_pubmed_doi": "10.1126/science.1093466",
  "rcsb_pubmed_abstract_text": "Resistin, founding member of the resistin-like molecule (RELM) hormone family, is secreted selectively from adipocytes and induces liver-specific antagonism of insulin action, thus providing a potential molecular link between obesity and diabetes. Crystal structures of resistin and RELMbeta reveal an unusual multimeric structure. Each protomer comprises a carboxy-terminal disulfide-rich beta-sandwich \"head\" domain and an amino-terminal alpha-helical \"tail\" segment. The alpha-helical segments associate to form three-stranded coiled coils, and surface-exposed interchain disulfide linkages mediate the formation of tail-to-tail hexamers. Analysis of serum samples shows that resistin circulates in two distinct assembly states, likely corresponding to hexamers and trimers. Infusion of a resistin mutant, lacking the intertrimer disulfide bonds, in pancreatic-insulin clamp studies reveals substantially more potent effects on hepatic insulin sensitivity than those observed with wild-type resistin. This result suggests that processing of the intertrimer

```

```
disulfide bonds may reflect an obligatory step toward activation
." ,
"rcsb_pubmed_affiliation_info": [
  "Department of Biochemistry and Molecular Biophysics, Columbia
  University, New York, NY 10032, USA."
],
"rcsb_pubmed_mesh_descriptors": [
  "Molecular Weight",
  "Humans",
  "Liver",
  "Disulfides",
  "Molecular Sequence Data",
  "Crystallography, X-Ray",
  "Adipocytes",
  "Glucose",
  "Adiponectin",
  "Proteins",
  "Protein Structure, Quaternary",
  "Resistin",
  "Protein Structure, Tertiary",
  "Amino Acid Sequence",
  "Cell Line",
  "Protein Structure, Secondary",
  "Intercellular Signaling Peptides and Proteins",
  "Crystallization",
  "Insulin Resistance",
  "Hormones, Ectopic",
  "Insulin",
  "Protein Folding",
  "Culture Media, Conditioned",
  "Animals",
  "Mice",
  "Mutation"
],
"rcsb_pubmed_mesh_descriptors_lineage": [
  {
    "id": "E05.196.300",
    "name": "Crystallization",
    "depth": 3
  },
  ...
  {
    "id": "E05.196.309.742",
    "name": "X-Ray Diffraction",
```

```

    "depth": 4
  },
}

```

mmCIF

mmCIF is a file format for representing macromolecular structural data with flexible and extensible tag-values. It is an alternative to the Protein Data Bank (PDB) format and is currently the default format used by the Protein Data Bank since 2014 (PDBx/mmCIF), representing data in key-value or tabular forms, being much easier to parse than the record-oriented PDB format. The columns no longer have fixed widths and blank values, being substituted with points or interrogation marks. Preceding the rows with the values, we have a list of identifiers and metadata for the rows, useful for parsers and allowing to add more columns.

Example of the identifiers and the first row of the `_struct_conf` (corresponding to the HELIX records in a `pdb` file):

```

_struct_conf.conf_type_id
_struct_conf.id
_struct_conf.pdbx_PDB_helix_id
_struct_conf.beg_label_comp_id
_struct_conf.beg_label_asym_id
_struct_conf.beg_label_seq_id
_struct_conf.pdbx_beg_PDB_ins_code
_struct_conf.end_label_comp_id
_struct_conf.end_label_asym_id
_struct_conf.end_label_seq_id
_struct_conf.pdbx_end_PDB_ins_code
_struct_conf.beg_auth_comp_id
_struct_conf.beg_auth_asym_id
_struct_conf.beg_auth_seq_id
_struct_conf.end_auth_comp_id
_struct_conf.end_auth_asym_id
_struct_conf.end_auth_seq_id
_struct_conf.pdbx_PDB_helix_class
_struct_conf.details
_struct_conf.pdbx_PDB_helix_length
HELX_P HELX_P1 AA1 GLY A 34 ? LEU A 48 ? GLY A 34 LEU A 48 1 ? 15

```

The SHEET records are now split into several groups and if we want details about each strand we need to go to the `_pdbx_struct_sheet_hbond` group. Here is an example of the identifiers and of the first row:

```

_pdbx_struct_sheet_hbond.sheet_id
_pdbx_struct_sheet_hbond.range_id_1
_pdbx_struct_sheet_hbond.range_id_2
_pdbx_struct_sheet_hbond.range_1_label_atom_id
_pdbx_struct_sheet_hbond.range_1_label_comp_id
_pdbx_struct_sheet_hbond.range_1_label_asym_id
_pdbx_struct_sheet_hbond.range_1_label_seq_id

```

```

_pdbx_struct_sheet_hbond.range_1_PDB_ins_code
_pdbx_struct_sheet_hbond.range_1_auth_atom_id
_pdbx_struct_sheet_hbond.range_1_auth_comp_id
_pdbx_struct_sheet_hbond.range_1_auth_asym_id
_pdbx_struct_sheet_hbond.range_1_auth_seq_id
_pdbx_struct_sheet_hbond.range_2_label_atom_id
_pdbx_struct_sheet_hbond.range_2_label_comp_id
_pdbx_struct_sheet_hbond.range_2_label_asym_id
_pdbx_struct_sheet_hbond.range_2_label_seq_id
_pdbx_struct_sheet_hbond.range_2_PDB_ins_code
_pdbx_struct_sheet_hbond.range_2_auth_atom_id
_pdbx_struct_sheet_hbond.range_2_auth_comp_id
_pdbx_struct_sheet_hbond.range_2_auth_asym_id
_pdbx_struct_sheet_hbond.range_2_auth_seq_id
AA1 1 2 N GLU A 17 ? N GLU A 17 0 LYS A 25 ? 0 LYS A 25

```

For general details about sheets, we have the `_struct_sheet` group:

```

_struct_sheet.id
_struct_sheet.type
_struct_sheet.number_strands
_struct_sheet.details
AA1 ? 4 ?
AA2 ? 4 ?
AA3 ? 2 ?
AA4 ? 2 ?
AA5 ? 2 ?
AA6 ? 3 ?
AA7 ? 4 ?
AA8 ? 4 ?
AA9 ? 2 ?
AB1 ? 2 ?

```

Example of the identifiers and the first ten rows of the `_pdbx_poly_seq_scheme` (corresponding to the SEQRES records in a pbd file):

```

_pdbx_poly_seq_scheme.asym_id
_pdbx_poly_seq_scheme.entity_id
_pdbx_poly_seq_scheme.seq_id
_pdbx_poly_seq_scheme.mon_id
_pdbx_poly_seq_scheme.ndb_seq_num
_pdbx_poly_seq_scheme.pdb_seq_num
_pdbx_poly_seq_scheme.auth_seq_num
_pdbx_poly_seq_scheme.pdb_mon_id
_pdbx_poly_seq_scheme.auth_mon_id
_pdbx_poly_seq_scheme.pdb_strand_id
_pdbx_poly_seq_scheme.pdb_ins_code
_pdbx_poly_seq_scheme.hetero
A 1 1 MET 1 1 ? ? ? A . n
A 1 2 GLN 2 2 ? ? ? A . n
A 1 3 GLY 3 3 ? ? ? A . n
A 1 4 SER 4 4 ? ? ? A . n
A 1 5 VAL 5 5 5 VAL VAL A . n

```



```
A 1 6   THR 6   6   6   THR THR A . n
A 1 7   GLU 7   7   7   GLU GLU A . n
A 1 8   PHE 8   8   8   PHE PHE A . n
A 1 9   LEU 9   9   9   LEU LEU A . n
A 1 10  LYS 10  10  10  LYS LYS A . n
```

Example of the identifiers and the first row of the `atom_site.group` (corresponding to the ATOM records in a pbd file):

```
_atom_site.group_PDB
_atom_site.id
_atom_site.type_symbol
_atom_site.label_atom_id
_atom_site.label_alt_id
_atom_site.label_comp_id
_atom_site.label_asym_id
_atom_site.label_entity_id
_atom_site.label_seq_id
_atom_site.pdbx_PDB_ins_code
_atom_site.Cartn_x
_atom_site.Cartn_y
_atom_site.Cartn_z
_atom_site.occupancy
_atom_site.B_iso_or_equiv
_atom_site.pdbx_formal_charge
_atom_site.auth_seq_id
_atom_site.auth_comp_id
_atom_site.auth_asym_id
_atom_site.auth_atom_id
_atom_site.pdbx_PDB_model_num
ATOM 1   N  N   . VAL A 1  5   ? 239.583 162.501 203.617 1.00 62.62  ? 5   VAL
  A  N   1
```

Converters

There are many online converter tools we can use to get the identifier of an entity in another repository. We will talk about a few of them.

biodbnet : db2db [39]

db2db is an application from the biological DataBase network (bioDBnet). It integrates many biological databases and handles all the conversions from one database identifier to another. Without this conversion, it would not be easy to search different databases for the same thing.

The current database update is from December 2020 and it has 211 distinct nodes and 758 edges, meaning databases and relations between them.

We present below an example of a request:

```
GET https://biodbnet-abcc.ncifcrf.gov/webServices/rest.php/
biodbnetRestApi.xml?method=db2db&format=row&input=genesymbol&inputValue
```

s=MYC,MTOR&outputs=geneid,affyid&taxonId=9606

and its reply:

```
<?xml version="1.0"?>
<response>
  <item>
    <InputValue>MYC</InputValue>
    <GeneID>4609</GeneID>
    <AffyID>1827_s_at [Chip: U95A]//202431_s_at [Chip: GeneProfilingArray]//202431
      _s_at [Chip: U133A_2]//ADXECRS.3198_s_at [Chip: Xcel]//ADXEC.556.C1_x_at [
        Chip: Xcel]//1936_s_at [Chip: U95A]//37724_at [Chip: U95Av2]//202431_s_at [
        Chip: HT_HG-U133A]//202431_s_at [Chip: U133A]//202431_PM_s_at [Chip:
        U133_Plus_PM]//202431_s_at [Chip: Focus]//1936_s_at [Chip: U95Av2]//1973
        _s_at [Chip: U95A]//37724_at [Chip: U95A]//ADXECAD.20878_s_at [Chip: Xcel
        ]//1827_s_at [Chip: U95Av2]//11745021_a_at [Chip: HG-U219]//1973_s_at [
        Chip: U95Av2]//g12962934_3p_a_at [Chip: U133_X3P]//202431_s_at [Chip:
        U133_Plus_2]//11745021_a_at [Chip: PrimeView]//ADXEC.556.C1_at [Chip: Xcel
        ]//ADXECAD.15167_at [Chip: Xcel]</AffyID>
  </item>
  <item>
    <InputValue>MTOR</InputValue>
    <GeneID>2475</GeneID>
    <AffyID>11716133_a_at [Chip: HG-U219]//11716133_a_at [Chip: PrimeView]//202288
      _PM_at [Chip: U133_Plus_PM]//RC_H79143_at [Chip: Hu35KsubD]//202288_at [
        Chip: HT_HG-U133A]//40139_at [Chip: U95Av2]//11716134_a_at [Chip: HG-U219
        ]//267_at [Chip: U95Av2]//11716134_a_at [Chip: PrimeView]//215381_at [Chip:
        U133A]//215381_at [Chip: U133_Plus_2]//215381_PM_at [Chip: U133_Plus_PM
        ]//202288_at [Chip: GeneProfilingArray]//45997_at [Chip: U95B]//
        RC_N64398_at [Chip: Hu35KsubC]//11750567_a_at [Chip: HG-U219]//215381_at [
        Chip: HT_HG-U133A]//202288_at [Chip: U133A_2]//40139_at [Chip: U95A]//Hs
        .288569.0.S1_3p_x_at [Chip: U133_X3P]//11750567_a_at [Chip: PrimeView]//267
        _at [Chip: U95A]//215381_at [Chip: U133A_2]//202288_at [Chip: U133A
        ]//202288_at [Chip: U133_Plus_2]//ADXEC.12115.C1_at [Chip: Xcel]//
        g3282238_3p_at [Chip: U133_X3P]//215381_3p_at [Chip: U133_X3P]//215381_at [
        Chip: GeneProfilingArray]</AffyID>
  </item>
</response>
```

CTS - The Chemical Translation Service ²

We can use this service to translate to and from many identifiers, for instance:

Convert from one ID to another

<http://cts.fiehnlab.ucdavis.edu/rest/convert/{from}/{to}/{query}>

example of request:

<https://cts.fiehnlab.ucdavis.edu/rest/convert/InChIKey/Chemical%20Name/QNAYBMKLOCPYGJ-REOHCLBHSA-N>

example of the reply:

²<http://cts.fiehnlab.ucdavis.edu/>

```
[
  {
    "fromIdentifier": "InChIKey",
    "searchTerm": "QNAYBMKLOCPYGJ-REOHCLBHSA-N",
    "toIdentifier": "Chemical Name",
    "result":
      [
        "L-Alanine"
      ]
  }
]
```

We can get a list of valid source databases for conversion of ids:
<https://cts.fiehnlab.ucdavis.edu/rest/fromValues>

And a list of valid target databases for conversion of ids:
<https://cts.fiehnlab.ucdavis.edu/rest/toValues>

It can also be used to expand chemical formulas:
<https://cts.fiehnlab.ucdavis.edu/rest/expandformula/H2O>

```
{
  "formula": "H2O",
  "result": "HHO"
}
```

g:profiler [46]

g:profiler is part of the ELIXIR infrastructure. It provides a toolset to convert between gene identifiers, to find biological categories, and to map genes with orthologies.

g:Convert is the converter available at g:profiler. We can use it to convert between various protein, gene, and other identifier types. It relies on Ensembl as a primary data source and is able to identify what the input is without the user having to indicate.

The REST API is available at this endpoint: <https://biit.cs.ut.ee/gprofile> r/api/convert/convert/ and the input must be a json object, specifying the target, the organism and a query with identifier(s).

Example of a request to the g:convert API, in Python:

```
import requests

r = requests.post(
    url='https://biit.cs.ut.ee/gprofiler/api/convert/convert/',
    json={
        'organism':'hsapiens',
        'target':'UCSC',
        'query':['CASQ2', "CASQ1", "GSTO1", "DMD", "GSTM2"],
    }
)
```

and the response data is parsed with `r.json()['result']` to obtain:

```
{
  "converted": "uc001efx.5",
  "description": "calsequestrin 2 [Source:HGNC Symbol;Acc:HGNC:1513]",
  "incoming": "CASQ2",
  "n_converted": 1,
  "n_incoming": 1,
  "name": "CASQ2",
  "namespaces": "ENTREZGENE, HGNC, UNIPROT_GN, WIKIGENE",
  "query": "query_1"
},
...
{
  "converted": "uc057jbj.1",
  "description": "glutathione S-transferase mu 2 [Source:HGNC Symbol;Acc:HGNC:4634]",
  "incoming": "GSTM2",
  "n_converted": 14,
  "n_incoming": 5,
  "name": "GSTM2",
  "namespaces": "ENTREZGENE, HGNC, UNIPROT_GN, WIKIGENE",
  "query": "query_1"
},
{
  "converted": "uc057jbr.1",
  "description": "glutathione S-transferase mu 2 [Source:HGNC Symbol;Acc:HGNC:4634]",
  "incoming": "GSTM2",
  "n_converted": 15,
  "n_incoming": 5,
  "name": "GSTM2",
  "namespaces": "ENTREZGENE, HGNC, UNIPROT_GN, WIKIGENE",
  "query": "query_1"
}
```

2.2 Tools

Here we will talk about cheminformatics tools and some wrappers that facilitate and simplify access to these API.

PaDEL-Descriptor [64]

PaDEL-Descriptor is a software to calculate molecular descriptors. A descriptor describe quantitatively the molecules' physical and chemical information. A chemical fingerprint is like if a molecule left a trace. Through mass- or x-ray-spectroscopy, unique patterns are discovered in samples. it currently calculates 1875 different descriptors (1444 being 1D/2D and 431 3D) and more than 6000 fingerprint parameters.

PaDEL is available as a graphic interface, command-line and extensions for both RapidMiner and Knime. For the calculation it uses The Chemistry Development Kit (CDK), a library for bio- and cheminformatics written in Java.

PaDELPy ³

PaDELPy is a Python wrapper for the PaDEL-Descriptor Calculation Software that provides methods to access PaDEL from within Python. We can use its two major functions *from_md1* and *from_smiles* to obtain descriptors/fingerprints from an MDL MolFile or from a SMILES string, respectively, as well as other methods for more controlled input and output possibilities.

The lastest release is 0.1.10 from July 12th 2017.

It is installed with

```
pip install padelpy
```

PubChemPy ⁴

PubChemPy is a Python wrapper that provides methods to access the PubChem PUG REST API from within Python. We can access individual properties as well as full records of both compounds and substances in the PubChem repository. We can also get substructures or superstructures of those presented as inputs.

The lastest release is 1.0.4 from April 11th 2017.

It is installed with

```
pip install pubchempy
```

BioPython ⁵

BioPython is a collection of free Python libraries and applications, including wrappers to access Bioinformatics repositories like NCBI Entrez, GenBank, Kegg, PDB and Uniprot among others; parsers for various Bioinformatics file formats such as

³<https://github.com/ECRL/PaDELPy>

⁴<https://pubchempy.readthedocs.io/en/latest/>

⁵<https://biopython.org/>

BLAST, FASTA, Genbank, and others; and parsers for XML returns from the requests to those services. It also includes data mining algorithms as Logistic Regression, Markov Model, Max Entropy, Naïve Bayes and KNN.

It is installed with

```
pip install biopython
```

g:profiler client⁶

gprofiler-official is the official Python client to access the g:Profiler toolkit. It currently supports 98 different namespaces for enrichment analysis of functional terms, conversion between identifiers and mapping orthologies.

It is installed with

```
pip install gprofiler-official
```

The latest release is 1.0.0 from April 2th 2019.

2.3 Related Work

For broader ILP-based applications please see section 3.2.

Gene prediction and analysis tools presented in [34], [41] and [45] give a good perspective on the data mining approaches, tools to use like Weka and DeepLearning4j, and can introduce to some of the repositories we will also access. They can also provide useful info and insights regarding data preparation and results evaluation in genomics studies.

The "Portal para Enriquecimento de Informação Genômica e Proteômica"[54] uses proteomics information but only for clustering with no enrichment, whereas "A Toolbox for Genetic Studies"[45] does use proteomic and metabolomic data to predict cancer with SVM and C4.5.

Some of these projects mention the necessity to add other repositories and other data mining techniques. A good source for obtaining datasets, Pisces [57], is referenced on [45].

However, generally, data enrichment concerns web-based analysis tools [60]. Some of the available tools include:

- Enrichr⁷ works with libraries of genes that we can download, enrich and upload to analyse the enrichment;
- DAVID⁸ also has a huge integrated knowledge base of annotated genes and proteins;
- Panther⁹ is a classification system for proteins;
- g:Profiler¹⁰ classifies gene, functionally, through a statistical enrichment and also has an ID conversion tool;

⁶<https://pypi.org/project/gprofiler-official/>

⁷<https://maayanlab.cloud/Enrichr>

⁸<https://david.ncifcrf.gov/>

⁹<http://pantherdb.org>

¹⁰<https://biit.cs.ut.ee/gprofiler>

- Bioconductor¹¹ does gene expression functional enrichment analysis

An interesting approach is presented in [60], using a statistical test to rank categories in sets of over- or under-represented items.

Generally, the most used bioinformatics approaches include fuzzy clustering and k-means for gene expression and cancer subtypes grouping, protein classification with SVM, and decision trees. In cheminformatics, k-means is also very employed in subdividing large collections into smaller groups with some similarity and C4.5 is used to determine the toxicity or drug-like properties of compounds based on their molecular structure and important physical characteristics.

2.4 Summary

This chapter outlined the main concepts in Bioinformatics and Cheminformatics, including areas of application, specific tools and representation formats.

We have also presented important database repositories and their API, which will be important in developing this project as well as useful proxies, called wrappers, to access these REST API.

Finally, we discussed some of the approaches followed in related works in this area.

¹¹<https://bioconductor.org>

Chapter 3

Data Mining and Machine Learning Background

In this chapter we will discuss the state of the art in data mining and its applications pertaining to this project, including ILP concepts, notations, and techniques and also some useful data mining tools.

3.1 Relational Data Mining and Machine Learning

In the past, statistics were used to find patterns and to derive models. Nowadays, the amount of available data that needs to be searched is overwhelming, and every day more data is being produced and added to the public domain.

Data mining came to aid in that search for pattern and insights. It can handle vast quantities of data fast and efficiently. Its aim has been mainly to obtain knowledge and to make predictions.

Machine learning brought the ability to learn. In the same situations, it can evaluate the outcome of different applications.

Attributes

The available data is stored with attributes representing information about a certain feature. There are several types of attributes:

- Numeric or continuous attributes measure numbers. Discrete numbers, both real and integer-valued.
- Nominal, enumerated, discrete or categorical attributes correspond to a pre-determined, finite set of possibilities. Boolean attributes are one of the examples. However, if we have any boolean attribute, the problem is a mixed-attribute problem instead of a numeric-attribute one [63].
- Ordinal attributes allow sorting.
- Interval attributes are ordinal attributes with equal distance between the possible values.
- Ratio attributes are those that automatically have a reference (or zero) value.

Usually, only numeric, nominal and ordinal attributes are used in data mining, whereas machine learning can use more information about attributes [63].

We can use the algorithms for nominal attributes with numerical attributes if we can discretise them first.

Phases

In 1999 a new methodology for data mining was presented that described a process model to be followed in data mining, called CRISP-DM [51]. It is currently the most used model [8] and an open standard. Initially, before its public disclosure, five companies developed a project, with funding from the European Union, that grew into a consortium of over 200 [8].

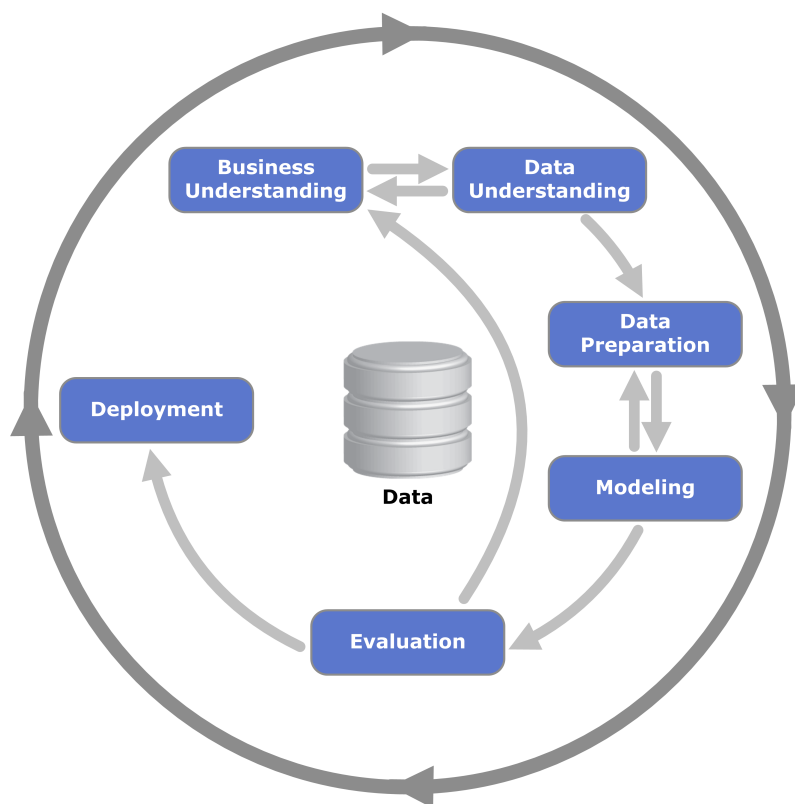


Figure 3.1: CRISP-DM diagram. Source: [56]

It consists of six major phases in a loop (Figure 3.1):

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment

Data Preparation, Modelling and Evaluation constitute a bigger phase that will be looped through until a satisfactory outcome is reached.

We will talk in detail about these three and the presentation of the results. In data mining, they are usually called pre-processing, mining, evaluation and knowledge representation.

Pre-processing

In the pre-processing phase, we have four tasks. First, we need to select which data to access. Then we need to clean the dataset.

This cleansing consists of removing noise, duplicate values, inconsistent data, and deciding what to do with missing values in the attributes. Other things to look for are inaccurate values and stale data. Stale data are values that can become incorrect over time or which can have an expiration date due to the volatility of the attribute.

Next, we have the integration of data from several sources. This project accesses different repositories to fetch data about the same instances and, therefore, integration is needed. Finally, the data is formatted to be better treated in the next phases. The attribute type can be modified.

Modelling

After the pre-processing, we apply one more data mining technique to extract the required knowledge.

Evaluation

After the pre-processing and data mining, we will evaluate the results. We will discuss it in the metrics section (section3.1).

LinearRegression

```
- 0.000 * wolf_energy  
+ 0.001 * sheep_vision_radius  
+ 0.000 * sheep_energy  
+ 0.024 * wolf_vision_radius  
+ 0.001 * sheep_count  
- 0.001 * comm_radius  
- 0.425
```

Figure 3.2: RapidMiner Linear Regression output example.

Knowledge representation

When we are satisfied with the results of the previous phase, we need to show the results. There are several ways of representing them.

We can present it in a table fashion, with each line being an element and the attributes in the columns. Each cell represents the value a given element has for a certain attribute.

Another way is to use linear models, presenting the values of the linear combination of the attributes multiplied by their weights (Figure 3.2).

Another efficient way of representing the results is by using decision trees (Figure 3.3b), as the product of a divide and conquer approach.

An alternative to decision trees are rules as an if... then clause (Figure 3.3a).

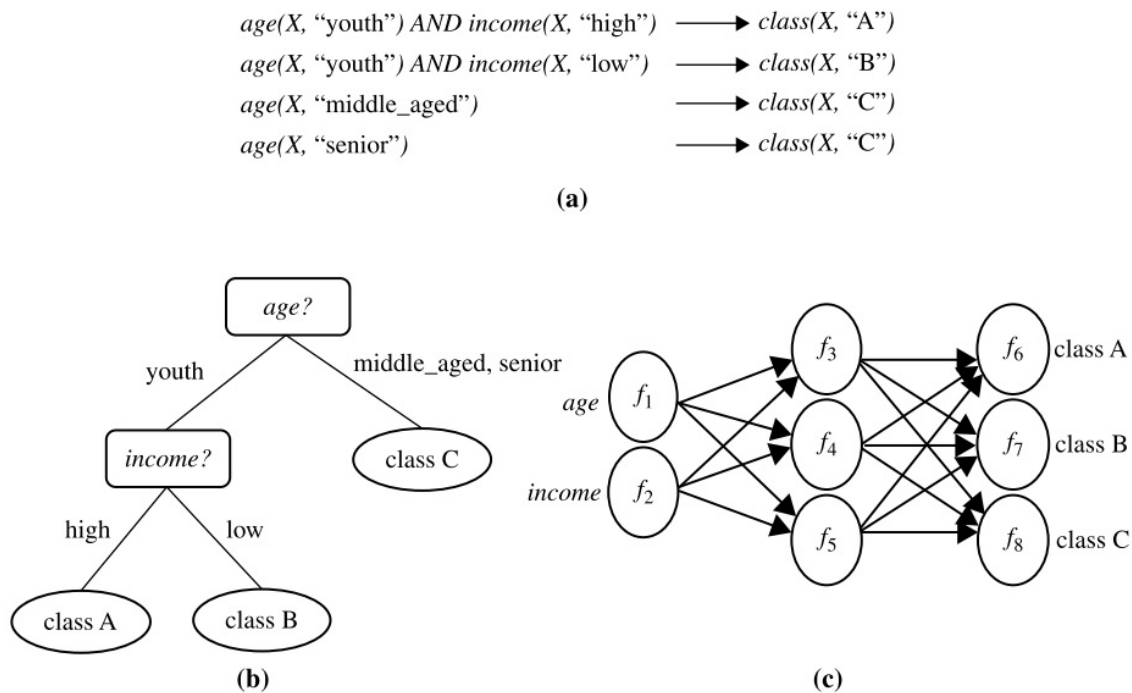


Figure 3.3: Examples of knowledge representation. Source: [63]

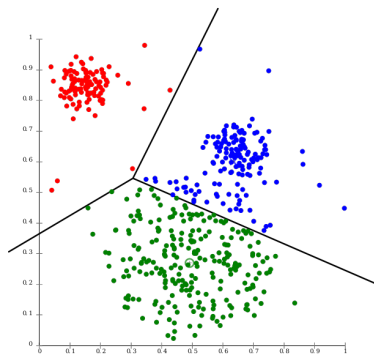
We can also present the decisions in a neural network manner (Figure 3.3c) if it is easily understandable.

If it is the case of a clustering solution, then the best representation is to use clusters. These can take different forms:

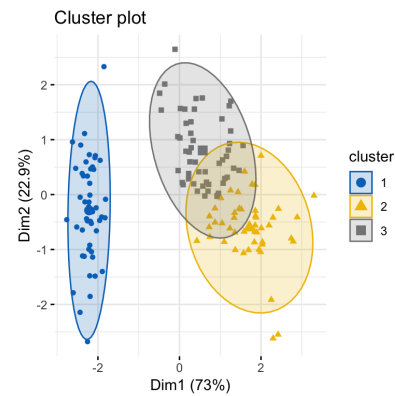
- A graphic with partition lines clearly separating one cluster from the others. Figure 3.4a
- Venn diagrams with the elements belonging to each cluster inside. Figure 3.4b
- A Dendrogram. It is a representation that looks like the roots of a tree with the height of each pair of roots representing the inverse similarity between them. Figure 3.5a
- A Distance matrix is a 2D colour-graded matrix showing the distance between the elements. Figure 3.5b

Overfitting

Overfitting occurs when a model is too much adapted to the training data, preventing the model from generalising and classifying new data correctly. In such cases, it is more likely that the model underwent memorisation than training. Overfitting tends to occur more often in quite elaborate models with a great number of parameters. We should be parsimonious in choosing how many descriptors to use. We should never employ more descriptors than are necessary to model a problem[20] as this leads to overfitting and lack of flexibility.

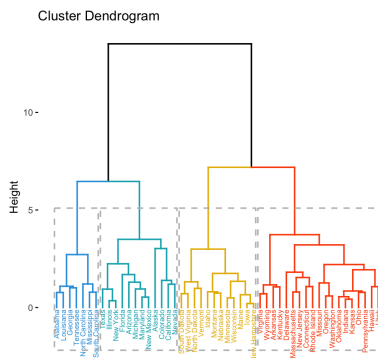


(a) Partition representation of clusters. Source: [62]

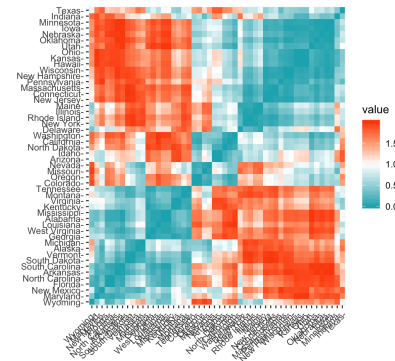


(b) Venn diagram representation of clusters. Source: [27]

Figure 3.4: Cluster representations



(a) Dendrogram representation of clusters. Source: [27]



(b) Distance matrix representation of clusters. Source: [27]

Figure 3.5: Cluster representations

Bias

During the whole process, we have to make assumptions. The assumptions we make can bias the modelling in a certain way.

Language Bias

When predicting the weather in Lapland, how many different words should we consider? Just one? Should we choose ten different types or the whole 180 words they have?

If the language we choose to define concepts imposes constraints on what concepts can be learned, we have language bias [63]. An example is nominal attributes in which we have to ponder if all possibilities are considered or if we should divide some into more subdivisions.

Search Bias

Most of the times it is not possible to search the whole data space and at the same time know for sure if the description found is the best one. So we have to make decisions and choose a search path, making the search procedure heuristic. Different search heuristics bias the search differently, and we cannot guarantee the result's optimality [63].

Overfitting-Avoidance Bias

To avoid overfitting, we may choose to use fewer features to solve the problem and bias the process somehow. Moreover, if we start with simple descriptions and proceed to more elaborate ones, it would bias the search favouring simpler concept descriptions [63].

Data Mining Techniques

Now we will discuss basic data mining concepts and techniques to provide fundamental prior knowledge needed to understand the project. We try to understand how they can be used in this project with different approaches and algorithms.

Classification

The classification, is part of what is called supervised learning, previews the outcome or class, based on the attributes, i.e., find associations between one or more attributes and the outcome in the training data. Those associations will be used to predict the outcome when applying the model to (new) test data. The outcome is viewed as the linear combination of the attributes, having weights as coefficients. To find those weights, we have to solve that kind of linear equation, known as the regression equation, from the training data outcomes.

Association

The association rules establish associations within attributes or between attributes and classes. It is not suited for prediction but can be used to find attributes that belong together, meaning that if one is present, the other(s) must also be. The association rules can be viewed as a type of functional dependency between the attributes in a database relation.

Regression

Regression implies finding a mathematical curve that explains the data and using it for prediction. The most important types of regression are linear and logistic regressions.

Clustering

Clustering, a type of unsupervised learning, is about aggregating elements into groups, based on their similarity. When assigning an element to a group, the objective is to maximise the similarities within each cluster and minimise the similarities between different clusters' elements.

Classification Algorithms

We have many algorithms to classify the data. Some of them are described next. It is recommended to try the simplest ones first as many times they work just as good as the more complicated ones [63].

1R

1R, meaning one rule, is a very simple algorithm. Each attribute is tested according to rules, and each different result produces a new branch, where another attribute will be tested until we have no more attributes to be tested. We count the sum of the different outcomes for each initial test. The chosen outcome of that test will be the outcome that appears more than the others.

k-nearest neighbours

This algorithm is very slow, because it has to search the entire set for each instance set ($\mathcal{O}n^2$ complexity). It performs badly in the presence of noise because for each test instance the class is defined by its sole nearest neighbour without averaging. It can be used in the classification of numerical attributes.

The algorithm assigns each element to the class the majority of elements in the k-nearest neighbourhood belong to. We define the parameter k , and it has to be an odd number to guarantee there is never a draw. In Figure 3.6, we see an example where we are trying to classify the blue element. If we choose $k=1$, the element will be classified as red. If we choose $k=3$, it will also fall into the red class because of three nearest elements, two are red. However, if we choose $k=5$, then the element will be classified as green because there are three green elements in the neighbourhood and only two are red.

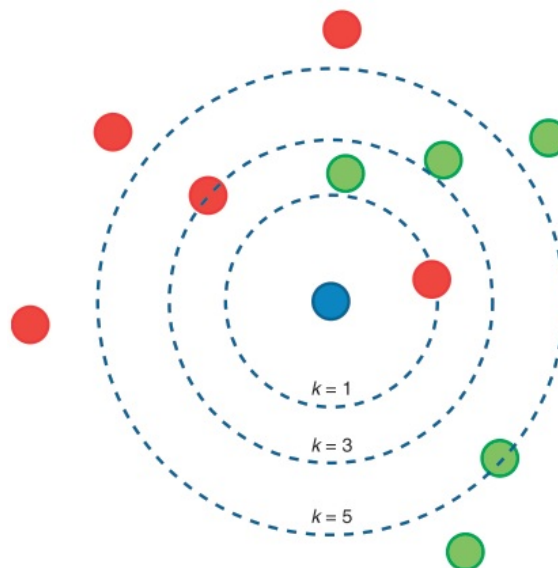


Figure 3.6: K-nearest neighbours example. Source: [38]

Naïve Bayes Classifier and Networks

The Naïve Bayes Classifier is a type of statistical modelling algorithm that learns conditional probability of each attribute given the class from training data.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \quad (3.1)$$

It is based on the Bayes' theorem (Equation 3.1) that calculates an event's probability, based on prior knowledge. Each attribute is considered independent from the others, and the numeric attributes must follow some distribution. Otherwise, it will not work. This algorithm is very good in prediction tasks and is widely used on document classification, being very fast and accurate in this domain. The existence of each word is regarded as a boolean attribute [63].

This classifier is viewed as a constrained form of a general Bayesian network, which is more complex and harder to compute. Bayesian networks are directed acyclic graphs and overcome the assumption of probabilistic independence between attributes given the class. They enable the "efficient and effective representation of the joint probability distribution across a set of random variables" [14].

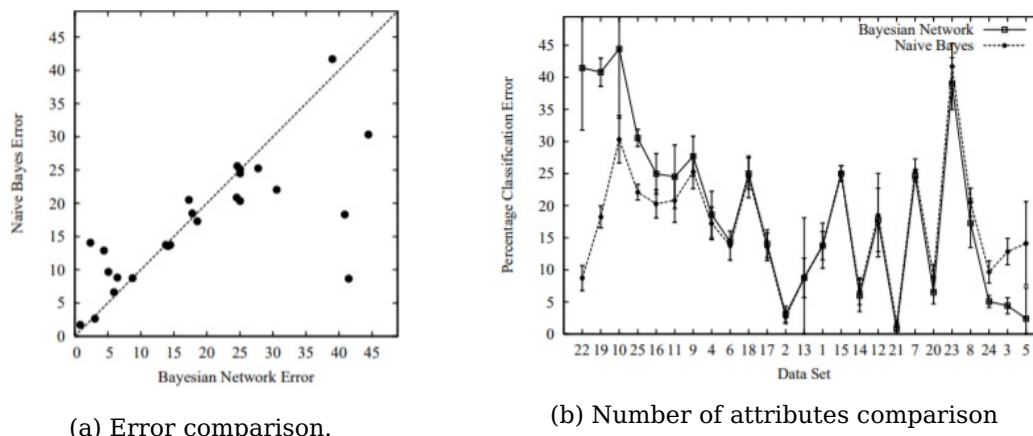


Figure 3.7: Comparison of Bayes networks classifiers. Source: [14]

In Figure 3.7, we see a comparison between these two Bayes network classifiers in a test with 25 different datasets. In the plot on the left (Figure 3.7a), each dataset is depicted by a point. The error ratio according to unsupervised Bayesian networks is the abscissa, and the error ratio of naïve Bayes is the ordinate. We can conclude that unconditional Bayesian networks perform better above the diagonal line and naïve Bayes performs better below. Unconditional Bayesian networks performed poorly on those containing more than 15 attributes (Figure 3.7b) [14]. So we can derive that naïve Bayes performs better than an unconditional Bayes network with bigger numbers of attributes.

ID3 and C4.5 (Divide and Conquer)

Divide and conquer algorithms are based on decision trees constructed recursively, branching at each node, in a greedy way. The nodes that we should branch next are the ones that provide more info about the solution to the problem.

In 1986, Ross Quinlan developed ID3 (Iterative Dichotomiser 3). In 1993 he created C4.5, which is an improvement of ID3.

The info of each node is calculated using the concept of entropy (Equation 3.2 and 3.3).

$$p_n = \frac{\text{occurrence of } n}{\text{total of occurrences}} \quad (3.2)$$

$$\text{entropy}(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \dots - p_n \log p_n \quad (3.3)$$

The extreme cases are:

- If all children are the same then the info is minimum (0).
- If the count of each type of child is the same, then the info is maximum (1).

Decision trees work well with small datasets. They still perform well with larger ones and are easy to update with new features but they may increase in complexity and need branches to be cut out through pruning or other methods. They are not robust enough and are very sensitive to little variations in the training set [34].

Random Forest

A more efficient algorithm based on decision trees is the random forest. In this algorithm, we use a combination of several decision trees. In the end, we will choose the value that was predicted more times. In Figure 3.8, we see a forest of five different random trees, the class chosen in each is depicted by following the gold path. As four trees classify the instance as red against one that predicts it is green, in the end the algorithm will classify the instance as red (majority).

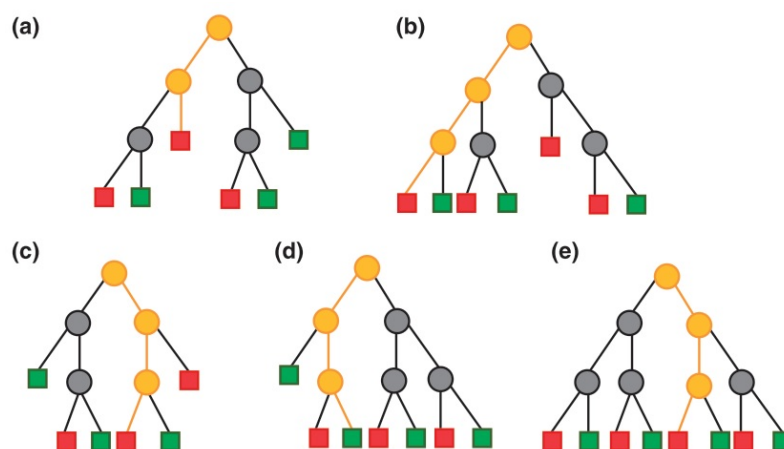


Figure 3.8: Random forest diagram. Source: [38]

Linear Regression

Used when the outcome and all attributes are numeric. The outcome is the linear combination of the sum of each attribute multiplied by a weight (coefficient).

We use the training data to solve these regression equations. We already have the attribute values and obtain their weights. As we can see in Equation 3.4, with the test data, we use those weights (w_i) applied to each attribute (a_i) to predict the numerical value of the outcome (x).

$$x = w_0 + w_1 a_1 + \dots + w_n a_n \quad (3.4)$$

This solution can be easily perceived, looking at a graphic representation of the linear combination equation.

Logistic Regression

In logistic regression, we separate one outcome from all the others for each class, obtaining a binary decision, modelled by an adaptation of the linear equation. What is important is that we obtain Equation 3.5 of a so-called hyperplane that separates the possibilities.

$$w_0 + w_1 a_1 + \dots + w_n a_n = 0 \quad (3.5)$$

Support Vector Machines

Support Vector Machines is a non-probabilistic linear binary classifier. This means it finds a maximum-margin hyperplane between two classes, a linear model that maximises the classes' separation. We can always subdivide the problem successively in x and $\neg x$ to obtain multiple classes.

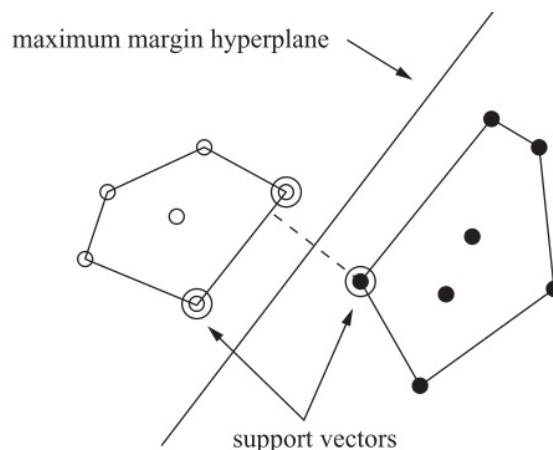


Figure 3.9: Support Vector Machines diagram. Source: [63]

The instances closest to the separation are called support vectors. We always have at least one support vector in each class, and usually more. "The set of support vectors uniquely defines the maximum-margin hyperplane" [63] for the learning problem

(Figure 3.9). We do not need any other training instances because they do not affect the hyperplane orientation or location. We can delete them [63].

If we have two attributes, we can write the hyperplane that separates the classes with Equation 3.6.

$$x = w_0 + w_1 a_1 + w_2 a_2 \quad (3.6)$$

and the maximum hyperplane would be written with Equation 3.7.

$$x = b + \sum_{i \text{ is support vector}} (\alpha_i y_i a(i) \cdot a) \quad (3.7)$$

Perceptron

Based on that notion of a hyperplane separating the instances, the perceptron appeared, which is a very simple learning rule to find one if it exists, as we can see in Algorithm 1.

Algorithm 1: Perceptron learning rule[63]

Result: weights

Set all weights to zero;

while not all instances in the training data are classified correctly **do**

for each instance I in the training data **do**

if I is classified incorrectly by the perceptron **then**

if I belongs to the first class **then**

 add it to the weight vector;

else

 subtract it from the weight vector;

end

else

end

end

end

The perceptron is the basis of artificial neural networks. A diagram of a generic perceptron is shown in Figure 3.10. In ANN, each neuron is a perceptron, with a function that processes the inputs, each pondered by a weight and the result adapted with a bias. If the value is above the activation threshold the signal from the perceptron will be fed to the next layer.

Winnow algorithm

Another algorithm that will find a separating hyperplane, if one exists, is the Winnow algorithm. It is very similar to the perceptron rule but, instead of altering the weight vector by adding, it multiplies it with a parameter α , specified by the user, greater than 1 [63]. We can also determine a threshold value for the instances to belong to a certain class (Equation 3.8).

$$w_0 + w_1 a_1 + \dots + w_n a_n > \theta \quad (3.8)$$

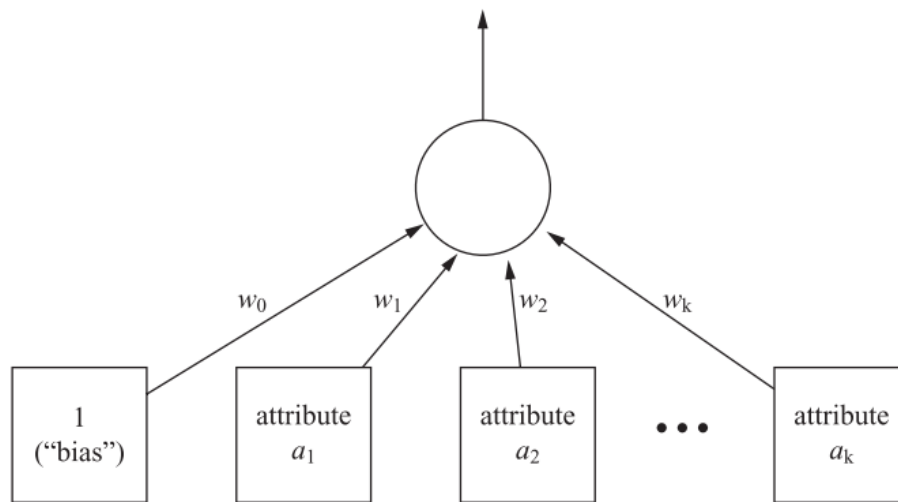


Figure 3.10: Diagram of a generic perceptron. Source: [63]

Artificial Neural Networks

Artificial Neural Networks (ANN) were inspired by the connections between the neurons in our brain and try to replicate what happens in the cerebral synapses with an intricate network of elements that perform functions.

These nodes in the ANN are also called neurons. Each connection has a weight associated with it and can also have a bias. The neurons process all the received input values, together with their weights and the bias and forward a single value which is the function of them all.

In an ANN we can have one or more hidden layers, each one composed by several of these neurons. Before the hidden layers we have the input layer to receive the data and pass it forward to be processed and at the end, the output layer receives the results and returns them.

There are several types of ANN, namely:

- Recurrent Neural Network (RNN): the connections between nodes form a directed graph along a temporal sequence, making it appropriate to predict temporal dynamic behaviour and for speech recognition.
- Convolutional Neural Network (CNN): fully connected networks (each neuron in one layer is connected to all neurons in the next layer), very successful in analysing images and detecting transaction frauds with cards.
- Graph Neural Network (GNN): offers a suitable means for node, edge and graph level prediction.
- Generative Adversarial Network (GAN): two neural networks fighting against each other in a zero-sum game, used to generate artificial representations of faces and more.

In Figure 3.11, we have a feed-forward ANN, because each layer feeds only the next one and no cycles are formed. RNN use backpropagation, meaning that it calculates the error and send it back to the earlier layers to correct the function.

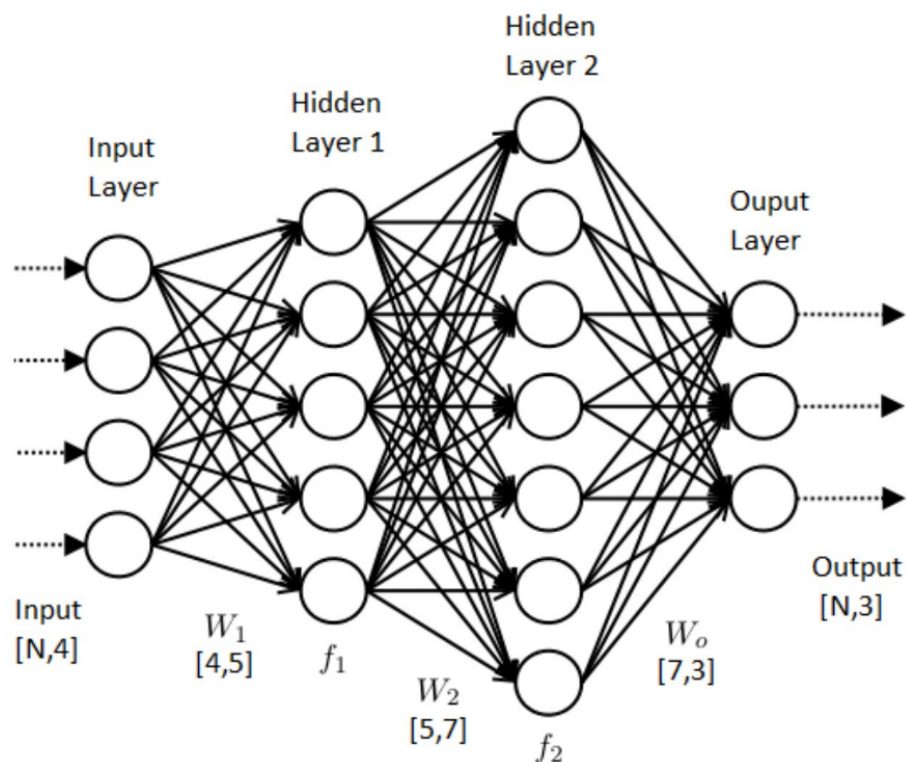


Figure 3.11: Diagram of a Artificial Neural Network. Source: VIASAT

Classification evaluation

How do we evaluate if the model is well trained and returns trustful values? We need to use quantifiable metrics that can measure the performance of the model.

First, We need to distinguish between two different concepts: accuracy and precision. Accuracy means all predictions are within a close range from the true value, whereas precision indicates how much each prediction is close to all the others. As we can see in Figure 3.12, we can have all combinations of these two concepts. It is preferable to have both high precision and high accuracy, but we may need just one of those or one more than the other for some problems.

However, which data should be used to evaluate the performance? When a solution to a problem is given as an example to a student, will we evaluate if he has grasped the knowledge by asking to solve that same example? Of course, we will not. Otherwise, we would be testing his memory. The same goes for data mining models. We can not predict the performance on the training set because it is not an independent test set [63].

We need to find a test set that had nothing to do with the classifier's training of the classifier, of which we know the outcomes, and use it to calculate the error rate of the classifier predictions.

Many times we have three different sets. A training data set used to train the model. Frequently, the bigger the training set, the more trustworthy the classifier. A validation set to improve the parameters of the classifier obtained. And a test set to

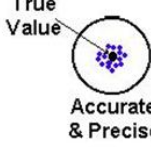


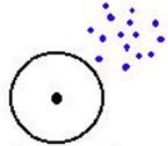
www.shmula.com		Accuracy	
		Accurate	Not Accurate
Precision	Precise	 <p>Accurate & Precise</p>	 <p>Not Accurate & Precise</p>
	Not Precise	 <p>Accurate & Not Precise</p>	 <p>Not Accurate & Not Precise</p>

Figure 3.12: Precision vs Accuracy. Source: [52]

calculate the error rate. The larger the test set, the more accurate the classifier will be [63].

Cross-validation

If we have a limited dataset, then we usually skip the validation set and use a minority of data to train the model and a majority to test it. Different methods on how to do this separation exists.

One of these methods is known as **holdout**. In this method, the dataset is split into two mutually exclusive subsets. We have to tune the ratio between them and choose a ratio that gives us good accuracy and good precision. We can start with 1/3 for training and 2/3 for testing and tweak from there. This method is usually more suited for larger quantities of data.

Another method is known as **k-fold**, or cross-validation k-fold. The dataset is randomly split up into k subsets of the same size. Iteratively, each subset will be used for training and the others for testing and its error rate is calculated. At the end of the k iterations, the global error rate is calculated by averaging the k error rates.

A method that increases the ratio of the training set while still keeping it independent is the **leave-one-out** cross-validation. This method is cross-validation executed n times, where n is how many instances exist in the dataset. Iteratively, we leave each instance out and train the classifier with the remaining $n-1$ instances. In each iteration, the testing is done on the instance that was left out and returning 1 if it is a hit or 0 if it is a miss. At the end of the n iterations, we calculate the mean of the n error rates and get the final error estimate.

While the other methods are all non-deterministic because the split is always random and we can get lucky in the split or not, the *leave-one out* method is deterministic because there is no stochastic sampling [63].

Bootstrap is another method and uses sampling with replacement. Whereas the previous methods would only have the same instance once in each subset, this method can use the same instance more than once in the same sampling group. If some instances can appear more than once, then there will be some instances that will not appear in the sampling group. As the sampling method is random, the probability of each of the n instances not appearing is the result of Equation 3.9.

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368 \quad (3.9)$$

We can infer that the test set contains circa 36.8% of the instances for a fairly large dataset. The training set has the remainder, around 63.2% (or 0.632). This number gives the full name of this method, *0.632 bootstrap* [63].

Confusion matrix

To calculate precision, accuracy and other metrics, we construct a 2x2 matrix known as a *confusion matrix*. One line has, for one of the possible outcomes, how many correct predictions and how many wrongful ones. The second line will have for the other outcome. We put confirmed values on the columns and predicted ones on the rows.

Classes	<i>buys_computer = yes</i>	<i>buys_computer = no</i>	Total	Recognition (%)
<i>buys_computer = yes</i>	6954	46	7000	99.34
<i>buys_computer = no</i>	412	2588	3000	86.27
Total	7366	2634	10,000	95.42

Figure 3.13: Extended confusion matrix example. Source: [19]

Figure 3.13, illustrates a confusion matrix extended with totals and the ratios of rightful predictions of each outcome and weighted global average.

	actual values	
	positive	negative
prediction of positive	TP	FP
prediction of negative	FN	TN

Figure 3.14: Confusion matrix.

The four cells of a confusion matrix (Figure 3.14) give us the following measurements:

- True Positives (TP): predicted positive and is true.
- False Positives (FP), also known as *type 1 error*, predicted positive and is false.
- True Negatives (TN) predicted negative and is true.

- False Negatives (FN), also known as *type 2 error*, predicted negative and is false.

Accuracy

Accuracy measures the ratio of the correct predictions, i.e. of all the cases, how many were correctly predicted (Equation 3.10).

$$accuracy = \frac{TP+TN}{Total} = \frac{TP+TN}{TP+TN+FP+FN} \quad (3.10)$$

The error rate is calculated with Equation 3.11:

$$error\ rate = 1 - accuracy. \quad (3.11)$$

Precision

Precision measures the ratio of the correct predictions of positives, i.e., of all the positives, how many were predicted to be positive (Equation 3.12).

$$precision = \frac{TP}{TP+FP} \quad (3.12)$$

Recall

Recall gives us the notion of how many of the predictions of positive are really positive (Equation 3.13).

$$recall = \frac{TP}{TP+FN} \quad (3.13)$$

F1-Score

The F1-score, also known as F-score and F-measure, helps measure Recall and Precision simultaneously (Equation 3.14). It is useful when we have models with low recall and high precision or vice versa.

$$F1 - score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \quad (3.14)$$

As we can see in Equation 3.14, F1-score weights equally both precision and recall. This will imply that even if either precision or recall is low, F1-score will also be low. With $F\beta$ -Score we can add some bias to that.

$F\beta$ -Score

$F\beta$ -Score is calculated with Equation 3.15 and is used when we don't want recall weighting the same as precision. For that matter, β represents how many times recall is more important than precision.

$$F\beta = (1 + \beta^2) * \frac{Precision * Recall}{\beta^2 * Precision + Recall} \quad (3.15)$$

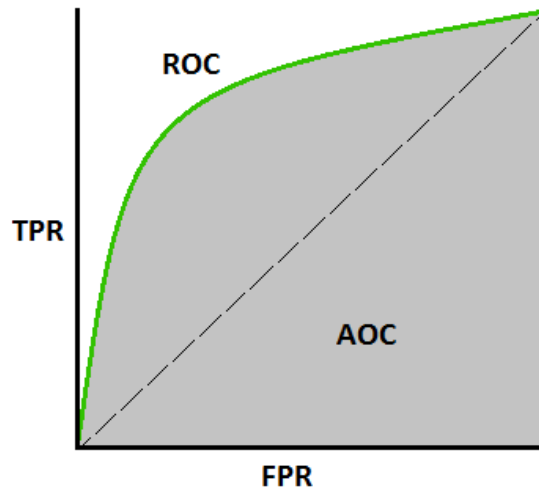


Figure 3.15: ROC and AUC. Source: [40]

ROC / AUC

Another possible measurement is to use the area under the ROC curve (AUC) (Figure 3.15). ROC is a probability curve of the TPR, true-positive rate (recall), as a function of the FPR, false-positive rate or the complement of the specificity. The specificity is the inverse of the recall (Equation 3.16). It measures the capability of the classifier to distinguish between classes. The higher the AUC, the better the model in making correct predictions [40].

$$specificity = \frac{TN}{TN + FP} \quad (3.16)$$

Baseline

After we get all the evaluation metrics, we must compare it with the results a basic solution would get to measure if our model is that good. The result of that basic model is a baseline, which we want to outperform. For both regression and classification problems, the baseline is designated Zero Rule, ZeroR or 0-R.

Clustering Algorithms

Now we will discuss some algorithms used to cluster the data elements.

Clustering algorithms can use different methods for the dataset partition. We can use distance-based algorithms employing a distance function that usually is Euclidean distance. This is also the basis of finding the nearest neighbours.

The categories into which the following methods are divided follow the division presented in [19].

Distance-based methods

Distance-based methods use the distances between one instance and the others to segment the data space, and the closest elements have the greatest influence.

Partitioning methods

Partitioning methods divide the data into k mutually exclusive groups such that each group must contain at least one object [19]. For convergence, the objects are, iteratively, reallocated among groups. These methods usually create all clusters at once [32].

These methods take a database with n elements and divide it into k partitions, or clusters, with the following constraints [41] :

- $k \leq \sqrt{N}$
- Each cluster holds at least one element.
- Each element belongs to just one cluster.

Most partitioning methods are distance-based, so they come up with clusters of spherical shape.

k-means

k-means is an iterative distance-based partitioning method and the classical clustering one. We begin by specifying how many clusters we want (k) and then choose k points in the data set that will be the centre of each of the k clusters. We attribute each element to each of those k clusters, based on the Euclidean distance to the nearest centre (Equation 3.17)

$$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (3.17)$$

After every element has been assigned to a cluster, we calculate each cluster's average value and that will be the centroid, or mean, of the next iteration. The algorithm converges when there is no change in any of the centroids from one iteration to the next.

kD-trees

kD-trees, short for k-dimensional trees, recursively partitions a k-dimensional space into two branches of a binary search tree. A variable i is calculated according to Equation 3.18. In each iteration, the algorithm assigns elements with the i th coordinate below a threshold to one branch and elements above it to the other. It is faster than k-means.

$$level \equiv i(\text{mod } k) \quad (3.18)$$

Expectation-Maximisation K-means is mostly based on Euclidean distances, outputting circular clusters. It will perform badly if the clusters have more variation in one dimension than the other. The Expectation-Maximisation algorithm overcomes this by adding a co-variance value to the calculation of the centroids. It is based on a probabilistic density statistical function and takes into account prior probability as a weighting factor.

The initialisation is similar to k-means, with the addition of the co-variance, then in each iteration we have a expectation step and a maximisation step.

In the expectation step it calculates the *a posteriori* probability for every class and every instance, with a formula similar to Bayes' conditional probability function. It returns a degree of belonging to a certain cluster.

The maximisation step maximises the expected likelihood from the previous step by finding parameters that will be used to calculate the distribution of the latent variables in the expectation step in the following iteration.

Hierarchical methods

Hierarchical methods decompose the data set in a tree-like fashion and have two different approaches. The *agglomerative* approach, also called the bottom-up approach, starts with individual leaves consecutively merging them in groups, until all the groups are merged into one at the tree's root. The *divisive* approach, also called the top-down approach, starts in the tree's root and iteratively, branches into smaller 'clusters' until reaching the leaves. Figure 3.16 explains the difference.

Hierarchical clustering methods can be distance-based or density-based [19].

This kind of method has a relatively low cost due to not being able to go back after each iteration. That is one major drawback so a pre-processing must be made beforehand to reduce the amount of data or of attributes to be used. Another possible enhancement is beginning with the agglomerative approach and later improve the results with the divisive approach [41].

Nearest neighbour chain

Nearest neighbour is an agglomerative hierarchical algorithm. It begins having n nodes (n being the sample's size). In each step, it joins the nodes with more similarity and ends when the number of wanted clusters is obtained.

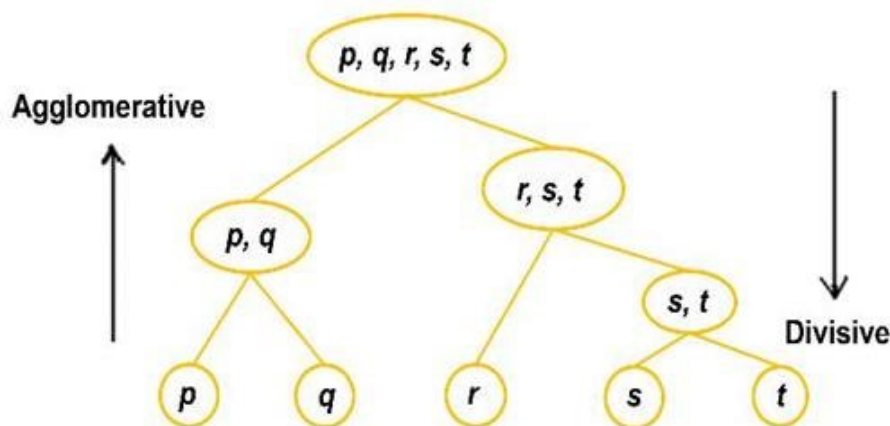


Figure 3.16: Diagram of the agglomerative and divisive approaches. Source: [16]

The nearest neighbour chain algorithm repeatedly follows a chain of clusters by transitivity from A to B to C... where each cluster is the nearest neighbour of the previous one.

Density-based methods

Density-based methods are not distance-based but based on the cluster's density. Distance-based methods tend to create spherical-shaped clusters. Density-based algorithms continue adding elements to the same cluster provided that the neighbourhood density is greater than a pre-specified value. The notion of a density neighbourhood is explained in Figure 3.17.

DBSCAN

DBSCAN uses the notion of minimum density. Each element in a cluster must have, within a given radius neighbourhood, at least a minimum pre-determined number of elements. Although it finds clusters with different shapes, it can not find clusters with different densities.

It begins identifying the k nearest neighbours of each point and the farthest k nearest neighbours. Then it calculates the average of all the farthest distances. It identifies, for each element in the dataset, the directly density-reachable ones using the minimum provided and classifies the points into either core or border points. Following this, it loops through all of the dataset and for the core elements it builds a new cluster based on the density reachable [32].

SSN

SSN differs from DBSCAN in defining the similarity between points by looking at the number of nearest neighbours that two points share. The density is determined by the sum of the similarities of the nearest neighbours of a point. [32]. It performs well in high-dimensional data and detects clusters of different shapes, densities and sizes [12].

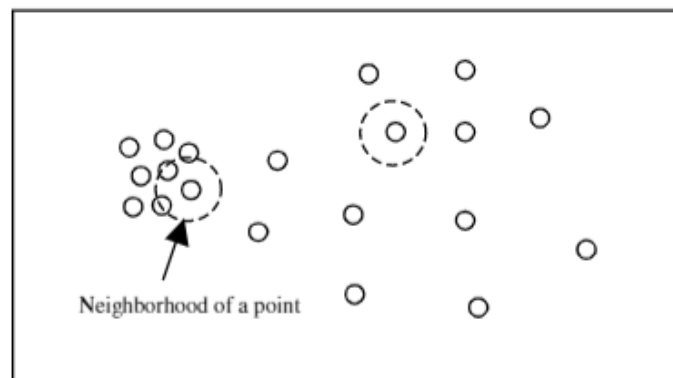


Figure 3.17: Density-based neighbourhoods. Source: [12]

Grid-based methods

Grid-based methods divide the data set into a limited number of cells and assign elements to the grid's suitable cell. Then calculate each cell's density. If a cell's density is below a given parameter, delete it. Finally, clusters are formed from adjacent groups of dense cells [32].

Because all operations are made on the grid, and the grid is not dependent on the amount of data, they are very fast [41].

Model-based methods

Model-based methods try to find a mathematical model that describes the data. More than often it assumes probabilistic distributions. They can have two principal approaches, the statistical approach and the neural network approach [32].

Fuzzy method

Fuzzy clustering, also known as soft clustering or soft k-means, is considered a hybrid method. With this algorithm, instances can belong to more than one cluster, hence the soft adjective. Each object has a set of membership coefficients that indicate the degree of belonging to a certain cluster.

Objects closer to the centre of a cluster may have a higher belonging coefficient than those in the cluster's limits. This measure is a number ranging from 0 to 1.

The approach is very similar to the one used by k-means:

- Choose how many clusters.
- Assign random coefficients to each instance.
- Repeat until the algorithm converges (the coefficients' change between two iterations is below a given threshold) :
 - Calculate the centroid of each cluster.
 - For each object, calculate its coefficients.

The centroid of a cluster is the average of all instances, weighted by their coefficient in that cluster, following Equation 3.19.

$$C_k = \frac{\sum_x w_k(x)^m x}{\sum_x w_k(x)^m} \quad (3.19)$$

where m is how fuzzy the cluster will be, with higher values making the cluster fuzzier.

If we know that the objects may belong to more than one cluster, we should choose this soft clustering approach. If we want exclusive clusters or if we know that there can not be overlapping clusters, we can try with a hard approach which is faster and computationally lighter [5].

Number of clusters

If we do not have a pre-determined number of clusters that we want, how do we know how many to ask the model for? If the model has a threshold option, like the density-based methods, we feed it that number. We can use a dendrogram to empirically try to determine a suitable number of clusters based on the similarities. We can also use a heuristic rule by cutting a dendrogram tree with maximum length [32].

Clustering Evaluation

The only thing we can really evaluate in these methods is whether the clusters we came up with prove themselves to be useful in the problem's context [63]. However, of the several methods available that try to evaluate the clustering model's performance, the Silhouette Coefficient, assumes that justified truth markers are available. Another option is to use a supervised learner on a related auxiliary task.

Silhouette Coefficient

This evaluation technique applies to any clustering method and gives a concise graphical representation of how well the instances have been classified.

The coefficient is a measure of the similarity between each instance and their cluster (cohesion) compared to other clusters (separation). It ranges between -1 and $+1$ and is calculated according to:

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ \frac{a(i)}{b(i)} - 1 & \text{if } a(i) > b(i) \end{cases}$$

with,

- $a(i)$: the average distance between instance i and the other instances in the same cluster.
- $b(i)$: the minimum average distance of instance i to any of the instances in any of the other clusters.

The highest the value, the better the instance is matched to its own cluster and worse to neighbouring clusters. If the majority of the instances are well matched to their own clusters, then the clustering is well done. If not, then we may have few or many clusters.

We can use both the Euclidean distance and the Manhattan distance.

The visualisation of the results is done by plotting a graph that combines the silhouette width of all the elements in the database with the average silhouette width of each cluster and the whole database's coefficient [41].

3.2 Inductive Logic Programming

In the last section, we presented the state of the art in propositional data mining, that deals with attribute-value elements in a database table. This paradigm has its drawbacks in solving certain types of problems like those related to molecular structures, maps and others.

Multi-relational data-mining is more suitable to address this category of issues. Several solutions include algorithms that cover all the tables in a database and graph representation learning.

This chapter will present a different approach, based on first-order logic clauses, that prove to be more effective than propositional methods when we have less specific training sets [29].

"Induction means reasoning from the specific to the general." [29]

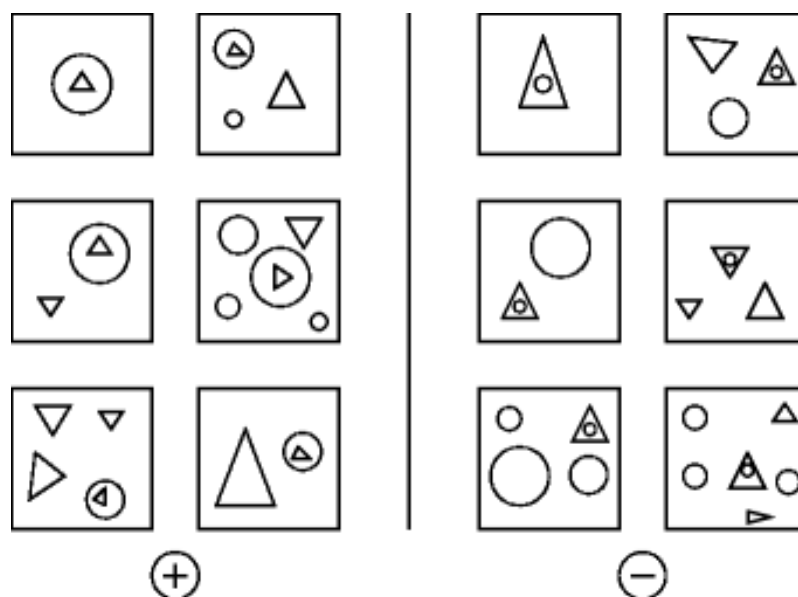


Figure 3.18: Positive and negative examples of triangles inside circles. Source: [10]

Inductive Concept Learning

The attributes of a table can be seen in propositional algorithms as the objects in inductive concept learning. The number and name of attributes are determined beforehand, and their values may or not be very well constrained. In inductive concept learning, we need to define a language to describe objects and to describe concepts, in order to be able to map:

concept \leftarrow condition between objects

Or

fact *if* hypothesis

We use two subsets of examples of facts. One holds positive examples (E^+) and the other, negative examples (E^-).

$$E = E^+ \cup E^-$$

From the facts, we try to reach a hypothesis.

If an object belongs to a concept, we say that “the concept description covers the object description” or that “the object description is covered by the concept description.”[29]

$covers(H, e)$, where H is the hypothesis and $e \in E$

Inductive Concept Learning with Background Knowledge

Difficult problems need previous knowledge of the world besides the facts, giving us more information on why those facts occur. This background knowledge “plays a central role in relationship learning” [29] and comes in a uniform representation of problem-specific data and general-purpose knowledge. With background knowledge, we achieve better models because the training accepts both positive and negative examples but also more rules that help them have a more suitable approach. Figure 3.18 tries to illustrate examples of sets with positive and negative examples. In that case, of triangles inside circles. The main idea behind inductive concept learning with background knowledge (B) is:

given E and B , find an H for a certain concept language L

Evaluation

There are several measurements to determine the performance of an ILP model:

Accuracy can be used to determine the percentage of objects correctly identified by the hypothesis.

$$\text{accuracy}(H) = \frac{\text{objects correctly identified by } H}{\text{total objects}} \quad (3.20)$$

Transparency tries to define to what extent the model is understandable by humans. It usually employs some measure of the final hypothesis's length, as the number of conditions on the final rule set or the number of bits used to encode it.

Statistical Significance is used to determine if it has a real distribution behind it or just some random assignment.

Information Content takes into account the difficulty of the problem. An example of how to calculate IS could be the following function:

$$IS(C_k) = \begin{cases} -\log(p(C)) + \log(p'(C)), & p'(C) \geq p(C) \\ \log(1 - p(C)) - \log(1 - p'(C)), & p'(C) < p(C) \end{cases}$$

where C is the concept, p is the prior probability and p' the value returned by the classifier.

Completeness – We say that the hypothesis is complete wrt. B and E if it covers every positive example.

$$H \text{ is complete if } covers(B, H, E^+) = E^+$$

Consistency - We say that the hypothesis is consistent wrt. B and E if it does not cover any negative example.

$$H \text{ is consistent if } covers(B, H, E^-) = \emptyset$$

First-order Logic

First-order logic is a group of formal notations and inferences used in computer science, using sentences with quantified variables. It uses Horn clauses in the reverse form to emphasise the induction aspect. A Horn clause is a logical formula used in logic programming: $(p \wedge q \wedge \dots \wedge t) \rightarrow u$

In ILP, background knowledge is represented as facts. These follow a fixed format as do hypotheses and the examples.

Background knowledge clauses are in the form:

```
grandfather(X,Y) :- father(X,Z), parent (Z,Y)
father(bob, alice)
```

And the set of clauses:

$$B = \{\text{mother(alice,david)}, \text{mother(alice,elias)}, \text{father(bob,alice)}\}$$

The positive examples fed to the training system are a non-empty set of Horn clauses whereas the negative set may be empty.

Example clauses are in the form:

```
grandfather(bob, charles)
```

And the two sets of examples can be:

$$E^+ = \{\text{grandfather}(\text{bob}, \text{charles}), \text{grandmother}(\text{bob}, \text{alice})\}$$

$$E^- = \{\text{grandfather}(\text{bob}, \text{alice})\}$$

An ILP problem can be defined as:

Given a set of examples E , a background knowledge B and a language L , find a hypothesis $H \subset L$ explaining the examples in E wrt. B .

Techniques

To generate the hypotheses from the example sets and background knowledge, several different techniques may be employed.

Bottom-up or generalisation

Generalisation begins with a very specific example and progressively groups more positive examples until it can no longer proceed without covering negative examples. In the end, it will deliver a hypothesis definition.

Least general generalisation

The least general generalisation is a way to compute a minimal complete generalisation set for a group of expressions. It has no redundant members but covers all possible generalisations of the examples given.

$$lgg(c_1, c_2)$$

Relative least general generalisation

The relative least general generalisation of two clauses c_1 and c_2 is their least general generalisation with respect to B .

$$rlgg(c_1, c_2)$$

Top-down or specialisation

The specialisation technique begins with a broader set of examples and tries to be more specific, using the refinement operator, to find a hypothesis definition:

Given a language L , a refinement operator ρ maps a clause c to a set of clauses $\rho(c)$ which are refinements of c :

$$\rho(c) = \{c' \mid c' \in L, c < c'\} \quad (3.21)$$

Covering Algorithm

The covering algorithm is a rule-based algorithm that builds hypotheses covering all positive examples but as few negative examples as possible. As long as there are positive examples left it will try to explain them through the rules in the background knowledge. Its implementation is explained in Algorithm 2.

Algorithm 2: Covering algorithm [1]**Repeat{**

- Learn one rule: This performs a general to specific search for a rule, which is highly accurate but has a lower coverage. It also learns a set of rules.
- Remove the data covered: The data covered in the algorithm is removed from the search space to make the process efficient.

}Until (best possible rule is generated)

Ontologies for ILP Background Knowledge

Ontologies are used to describe the concept language and object language.

An ontology is an accurate description of a collection of information in a specific field. It formats and constrains the meaning of a context and communicates and solves semantic inconsistencies. Ontologies normally consist of a collection of concepts with associations between them.

OWL

OWL, or Web Ontology Language, is a language for defining and representing ontologies on the WWW, including descriptions of classes, respective relationships and properties. It is intended for use by applications that require information processing, and it presents added vocabulary with an explicit definition.

GO

GO stands for Gene Ontology and represents the biological domain knowledge using a markup language, concerning:

- Molecular Function - activities that occur at the molecular level (e.g. catalysis)
- Cellular Component - where it operates (e.g. ribosome)
- Biological Process performed by various molecular functions (e.g. DNA repair)

Applications in Bioinformatics and Cheminformatics

Next, we will analyse some of the documented applications of ILP in bioinformatics and cheminformatics. These are old studies but work well as comparison examples.

Early diagnosis of rheumatic diseases

To diagnose rheumatic diseases at an early stage, Muggleton et al. (1992), according to [29], used attribute-value descriptions of patient data and a knowledge base provided by medical specialists, consisting of co-occurrences of symptoms. It achieved an accuracy of 72,9% when using background knowledge and only 62,8% without the

use of background knowledge, whereas "the relative information score of a classifier that always returns the prior probability distribution of diagnostic classes is zero (accuracy 34%)" [29].

Predict the secondary structure of proteins

The easiest way to predict a protein's geometry (secondary structure) is to verify if it applies to a specific known structure. The examples are relative to that specific structure, of positive examples with said structures and negative examples with different structures. Bear in mind that a protein is a sequence of amino acid residues and that its shape almost always determines its function.

Muggleton et al. (1992) and Sternberg et al. (1992), according to [29], provided examples with clauses with the format `structure(protein, position)`.

The background knowledge included which amino acids are common in which positions of certain proteins and known sequences of amino acids within proteins. It also incorporated pairs and triplets of amino acid residues that usually appear together when the protein has that structure and unary predicates of proteins' properties.

The training from the examples and the background knowledge a description was obtained with a set of rules of the properties, positions and sequences that induce the same format of the examples.

This approach obtained an accuracy of 81%, far better than neural networks had obtained until then (76%) and also of propositional approaches (73%).

SAR/QSAR of trimethoprim analogues

As with the prediction of structures, it is more efficient to predict a molecule's activity by comparing it with activities of known molecules.

According to [29], King et al. (1992) fed the model a set of examples with comparisons between the activity level of known compounds. The background knowledge had tuples with the radical that takes part in certain drugs and their structures. Other tuples included provided information about known radicals, as polarities, sizes and more.

The description induced by the training described the relative activity between molecules from structures, electron donors, flexibility and other properties.

In this case, it obtained a better accuracy (46%) than statistical models (42%) but lower than propositional approaches (54%).

The better performance of the propositional algorithms may rely upon the specificity in the problem. ILP can be better than propositional approaches if the training sets are less specific [29].

3.3 Tools

The following applications are some of the available tools specially dedicated to data mining. We use them to extract usable data from large sets, find patterns and correlations, extract knowledge, and make predictions.

Weka[13]

Weka stands for Waikato Environment for Knowledge Analysis and is a free, portable Java library for data mining, data analysis, and predictive modelling. It supports several standard data mining tasks, including data pre-processing, classification, clustering, feature selection, regression and association rules, as a class in a java package, like

- AssociationRules,
- BayesNet,
- BayesNetGenerator,
- DecisionTable,
- J48 (C4.5),
- LinearRegression,
- Logistic,
- MultilayerPerceptron,
- NaiveBayes,
- NeuralNetwork,
- OneR,
- RandomForest,
- Regression,
- RegressionByDiscretization,
- SimpleLinearRegression,
- SimpleLogistic,
- SupportVectorMachineModel,
- TreeModel
- ZeroR

The AssociationRules class includes various propositional algorithms.

Weka is as popular as RapidMiner as a data mining tool but lacks some specific advanced features [34]. By default, Weka uses 10-fold cross-validation, but we can define other values or select a split ratio if we prefer.

R and R-Studio

R is a programming language for machine learning statistical computing and analysis. It is popular in implementing regression, classification, and decision tree formation.

R-studio is an integrated development environment (IDE) for R and integrates with Tensorflow and other libraries. It also includes several external data mining packages and is possible to use Python with RStudio. The latest release is 1.4.1717 from June 1st 2021.

RapidMiner

RapidMiner Studio is an IDE for text mining, machine learning, predictive analytics, deep learning, data analysis and preparation.

It allows every data mining step, has a quite straightforward to use, intuitive, simple user-interface, loads and extracts information from unstructured data, has many available extensions and integrates with R and Python [34].

Aleph

Aleph stands for 'A Learning Engine for Proposing Hypotheses' and is the most used tool when it comes to Inductive Logic Programming. It is written in Prolog and it must be loaded inside a Prolog compiler or integrated in Knime or RapidMiner.

The Aleph system procedure works in four stages[53]:

- Selects an example to be generalised. Stops when there is none.
- Builds the most-specific-clause that covers the example, having the language restrictions into account.
- Finds a more general clause by searching a subset in the bottom clause with a best score.
- Adds the best scored clause to the current theory and removes all other examples, which are now considered redundant.

The background knowledge is in the form of Prolog clauses and the examples are ground facts.

The ARFF file format

An ARFF (Attribute-Relation File Format) file represents a series of instances sharing the same set of features. It is saved as an ASCII text file. This filetype was developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato to be used with the Weka machine learning software and is currently used by many tools [43].

Two parts form ARFF files. In the beginning, it has a header and then we have the Data information. It usually begins with authorship information in the form of comments. Comments appear after the % symbol.

The header includes the name of the relation and a list of the attributes with their types.

The ARFF header section contains the relation declaration and attribute declarations. All keywords are case insensitive, but string values and nominal attributes are case sensitive. The relation declaration is the first uncommented line in the ARFF file. A relation is declared in the following format:

```
@relation <relation-name>
```

where <relation-name> is a string. If the name includes spaces, the string must be quoted.

Then we have the declarations for each attribute in the relation, with its name and datatype, formatted like this:

```
@attribute <attribute-name> <datatype>
```

The datatype is one of these:

- numeric (real or integer numbers)
- <nominal-specification> (for nominal attributes)
- string
- date [<date-format>] (format is optional. default: "yyyy-MM-dd'T'HH:mm:ss" (ISO-8601 date and time format)).

For nominal attributes, we list the possible values (spaces imply quoting the string):

```
{<nominal-name1>, <nominal-name2>, <nominal-name3>, ...}
```

like the following example:

```
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The following is an example of a header on the standard IRIS dataset¹ (all the fragments were taken from the same source):

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%   (a) Creator: R.A. Fisher
%   (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%   (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength NUMERIC
@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petallength NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE class       {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The data section starts with a single line:

```
@data
```

Each instance has its own line, with attribute values separated by commas, in the same order of the declaration. The line ends with a carriage return. Any unknown value is represented by a question mark, as in:

```
4.4,?,1.5,?,Iris-setosa
```

¹https://waikato.github.io/weka-wiki/formats_and_processing/arff_developer/

The following is an example of a data section on the standard IRIS dataset:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

If there are many zero values, we can reduce the file size, by not representing them and use the sparse notation, with the other values explicitly identified by attribute number followed by value after a white space and adding the number of attributes after the last non-zero value:

ARFF:

```
@data
 0, X, 0, Y, "class A"
 0, 0, W, 0, "class B"
```

sparse ARFF:

```
@data
{1 X, 3 Y, 4 "class A"}
{2 W, 4 "class B"}
```

Scikit-learn [44]

Sci-kit Learn is a machine learning library for Python. It is used for data mining and data analysis and implements many algorithms for classification, regression and clustering, including support vector machines, random forests, k-means and DBSCAN.

The current version is 0.24.2 released in April 28th 2021.

It is installed through the Python package manager (pip) running the command:

```
pip install --upgrade scikit-learn
```


3.4 Summary

In this chapter, we have presented the state of the art in data mining and its applications pertaining to this project, complemented with an overview of ILP concepts, notations, and techniques with a special detailed description of the covering algorithm and some ontologies.

We have discussed different approaches and methods and how to evaluate their implementation and presented various tools that help us to achieve our goals.

Then, we described some of the applications of ILP on bioinformatics and cheminformatics together with the results of those applications and comparisons with the statistical and propositional approaches.

To finish this chapter, we presented the available Data Mining tools pertaining to this project.

Chapter 4

Implementation

This chapter will analyse the problem we are addressing and present our solution, together with implementation details, including the web development technologies used.

4.1 Requirements

Problem

Increasingly more complex problems are being addressed in life sciences. Acquiring all the data that may be related to the problem in question is paramount. Equally important is to know how the data is related to each other and to the problem itself. On the other hand, there are large amounts of data and information available on the Web. Researchers are already using Data Mining and Machine Learning as a valuable tool in their investigations. The usual procedure is to look for the information based on the inductive models.

Today, there are large amounts of data and a lot of information available on the net. It is not easy to integrate this vast amount of available information with propositional algorithms, that is, based on attributes, in the inductive process, that is, with supervised learning.

It happens, despite the great successes already achieved with the use of DM/ML.

The platform will produce outputs for both propositional and relational algorithms. We must enrich the training data to be used in the inductive process. Propositional algorithms are based on attributes and relational are based on first-order logic. With the two types of outputs one can compare the results using the two approaches.

Actors

The actors for this web application are just the *user* and the *Administrator*. An *User* can access all input pages and introduce terms and ids to obtain properties and save them to the database and/or generate the output files. An administrator can do everything a user can but also navigate in the Administration Area to explore the databases, import and export and also create other users.

We can also consider the repository and tools API as actors.

Functional Requirements

After several conferences with the client we came up with the functional requirements displayed on Table 4.1.

Table 4.1: Functional Requirements

Name	Description
Data Retrieval	The system must fetch data from public bioinformatics and cheminformatics repositories using API.
ID conversion	The system must convert identifiers from one repository to others.
Local Storage	The system must save the retrieved data locally.
Fetch Once	The system only accesses the repositories if the data is not yet available locally.
Prolog facts	The system must generate appropriate Prolog facts.
Save Prolog	The system must save the generated Prolog facts to .pl files.
Export Database	The system must be able to export database tables in several formats (including .csv).
CSV2ARFF	The system must be able to convert .csv files to .arff files.
Authentication	Only an authenticated user can access the platform.
Add Users	The system must allow the creation of other users and administrators.
Bulk Input	The system must allow the input of identifiers in bulk (via file input or copy/paste).
Padel	The system must allow the retrieval of PaDEL descriptors (with options).
Repository retrieval	The system must allow the input of individual identifiers on specific repositories.
Repository search	The system must allow the searching specific repositories for specified search terms.

Table 4.2: Restrictions

Name	Description
Delivery time	The application must be ready before October 20th 2021

Technical Requirements

From the same meetings, we devised Table 4.3, with technical requirements for the web application and the application's restrictions displayed on Table 4.2.

Table 4.3: Technical Requirements

Name	Description
Web applications	The system must be implemented with dynamic pages of web Applications
Availability	The system will be available 24 hours a day, 7 days a week, with minimal <i>downtime</i>
Usability	The system will be intuitive, easy to handle and to navigate and with a simple design.
Security	The system will be protected from unauthorized access and protected from attacks by third parties.
Speed	The system will be fast, with a short response time, offering a fluid experience to users.
Responsive	The system will be a responsive web site in order to adapt to different types of windows and/or devices.
Ethics	The system must comply with software development ethics laws, such as encrypting <i>authenticated users</i> passwords so that only they know their password.
Scalability	The system has to be built with the possibility of increasing the number of users in mind and adapting to changes in the ecosystem with the growth of <i>user base</i> .
Database	The system must make use of a database technology that allows the storage of big volumes of information.

4.2 Technologies

Considering the problem and the objectives, we created a web platform to obtain relevant information for Bioinformatics (particularly Genomics) and Cheminformatics problems. It fetches the data from public repositories with genomics, protein and chemical data and enriches it, with the help of ontologies and the PaDEL-descriptor. The new data is saved in files intended to be used by Prolog systems to be used in ILP algorithms. These systems give us a classification and evaluation of the performance.

The platform has a backend developed using python-based technologies.

Available Technologies

Next, we will discuss what technologies, languages and frameworks we have available to develop a web application to do the queries, display replies and in the backend, do requests to the repository, store the data in a database, thus facilitating analysis and making it faster.

Python

Python is a free, open-source, high-level interpreted programming language. It is very simple and one of the easiest to work with. Its popularity has to do with using it for different programming paradigms like object-oriented, imperative, procedural and imperative.

Having an enormous community contributes to it to have numerous libraries and frameworks.

There are many Django libraries and frameworks that make web development with Python more productive and fast.

Python provides data mining tools with dedicated libraries for data collection, data cleaning, data exploration, data modelling and data visualisation.

It is also very well served in automation frameworks and can scale very efficiently. The latest version is 3.9.6 from June 28th 2021.

Django¹

Django is a high-level web framework for Python. It is easy to work with complex databases, has a very easy to use database administration interface and has an excellent object-relational mapping solution in which we describe the database layout in Python code.

The latest release is 3.2.7 from September 1st 2021.

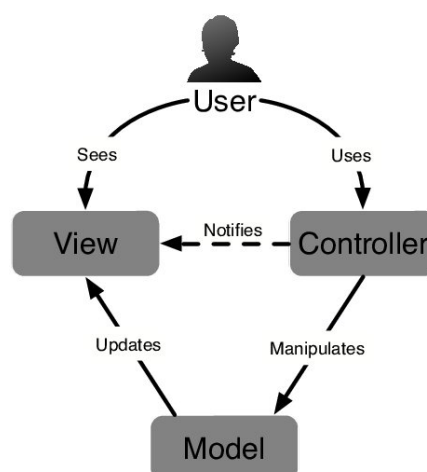


Figure 4.1: MVC architecture diagram. Source: [6]

¹<https://www.djangoproject.com/>

MVC architecture

MVC means model-view-controller. It is a software design pattern usually implemented on web applications to separate internal logic and databases from client-side visualisation code.

The model concerns the data, database management and the logic behind the application. The view deals with the visualisation, with what the user sees. The controller sends user requests to the model and notifies the view to receive new data from the model. Each component interacts with the others as we see in Figure 4.1.

MTV architecture

MTV means model-template-view and is the design pattern used in Django applications and is equivalent to the MVC [35].

Django handles the controller part of the MVC itself, and this design is sometimes also referred to as the Model-Template-View + Controller [35].

The model is the same as in MVC, acting as an interface between the data. It also deals with everything related to data access and validation.

The template is like the view in MVC. It controls what should be displayed and how.

The view acts as the controller in MVC. It deals with all business logic behind templates and connects the model and the template.

In Figure 4.2, we can see the MTV architecture of a Django application.

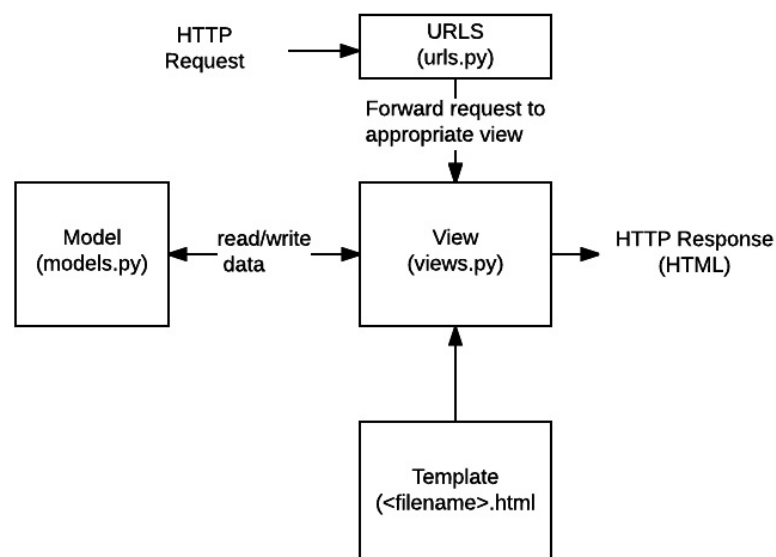


Figure 4.2: Django application architecture diagram. Source: [35]

React²

React is a JavaScript library for frontend development. With it, we can build complex user interfaces from small and isolated pieces of code called components.

React.JS makes it possible to easily display good looking and intelligible dashboards, graphics, graphs, and tables. It is done thanks to many external component libraries like

- Material-UI,
- Victory,
- React-vis,
- React D3,
- Chart.js
- ReGraph,
- ReCharts,
- ApexCharts,
- Nivo,
- React Google Charts

and many others.

Redux³

Redux is an open-source JavaScript library, commonly used with React for managing application state.

SQL

SQL is a declarative programming language used to design and manage data in a relational database management system (RDBMS). It is recommended when using structured data when there are relations between entities and variables. It has been a standard of the American National Standards Institute (ANSI) since 1986 and since the following year of the International Organization for Standardization (ISO).

The main purpose of SQL is to query data in a relational database. There are extensions to standard SQL that add procedural elements such as control-of-flow constructs. One of those extensions is PostgreSQL.

PostgreSQL⁴

PostgreSQL, also known as Postgres, is a powerful, open-source object-relational database system (RDBMS) that uses and extends the SQL language. The last stable release 13.4 was released on August 12th 2021.

It handles better big datasets than other database managers and can use Python as a procedural language.

The creators of Django "recommend PostgreSQL, which achieves a fine balance between cost, features, speed and stability".[22, 15]

²<https://reactjs.org/>

³<https://redux.js.org/api/api-reference>

⁴<https://www.postgresql.org/>

gunicorn⁵

Gunicorn is a Python Web Server Gateway Interface (WSGI) HTTP server. In Gunicorn, a master thread creates workers to handle requests but each worker is independent of the controller. The latter does not control how the workers handle the request. This is known as pre-work worker model.

nginx[47]

Nginx is used as a web server for static assets like as images, JavaScript and Cascade Style Sheets and also as a reverse proxy, which passes HTTP requests to a WSGI server like Gunicorn.

Docker[36]

Docker is a container-based tool to provide environments for development, deployment and production of web applications. It works as a virtual machine but it does not create a whole virtual operating system. It delivers the same Linux kernel to all phases of implementation and to all contributing entities, which enables an incredible performance improvement and reduction of the application's size.

4.3 Boilerplate

As a framework, we choose Django with React frontend. Django is easy to work with complex databases, has a very easy to use database administration interface and is developed in Python. React helps in displaying clean looking, easily re-adaptable interfaces and also adds possibilities to show charts, graphs and other analysis tools, if necessary.

To save the data already collected, thus facilitating the analysis and making it faster, we will have a PostgreSQL database.

First, we installed the Django framework, then the React library for the frontend. Then we dockerized the application with settings for a PostgreSQL database and usage of Nginx and Gunicorn servers. During development, several more packages and libraries were added, such as Redux and python wrappers.

The implementation of our application starts with the setting up of all the necessary technologies, the installation of modules and packages and the establishment of a boilerplate on which the rest of the development will build upon.

Django

To create a new Django application, first, we need to install the framework, which is easily done using the python pip package installer:

```
python -m pip install Django
```

Then we need to write the following command in the command line:

⁵<https://gunicorn.org/>

```
django-admin startproject <site\_name>
```

The app is created with the following structure:

```
site_name/
  manage.py
  site_name/
    __init__.py
    settings.py
    urls.py
    asgi.py
    wsgi.py
```

- *manage.py* is Django's command-line utility for administrative tasks and sets the `DJANGO_SETTINGS_MODULE` environment variable to point to the project's `settings.py` file.
- `settings.py` is the configuration file for the project.
- *urls.py* holds the URL declarations for the Django project, like a directory listing of endpoints. We can have other URL dispatchers in each project module and redirect from here to them after importing.
- *asgi.py* and *wsgi.py* are the entry-points for ASGI-compatible and WSGI-compatible web servers, respectively.

To help uniform the environment between different locations, we also add a `requirements.txt` file outside the project folder. This file holds the name of the modules and their versions that must be present for the Django project to run. Throughout the implementation phase, as modules were being added to the project, the file was updated. Then, to assure the modules are or will be installed, we run

```
pip install -r requirements.txt
```

This project has the main project folder like this:

```
mainapp/
  __init__.py
  local_settings.py
  settings.py
  urls.py
  asgi.py
  wsgi.py
```

There are also folders for each module, inside the project folder, alongside the `mainapp` folder:

```
django_app/
  converter/
  ensembl/
  genbank/
```

```
files/  
go/  
kegg/  
mainapp/  
output/  
padel/  
pubchem/  
reqs/  
static/  
users/  
utils/
```

The `files` folder holds the downloaded `.sdf`, `.mmcif` and other files. The `users` module deals with user authentication. To hold static files like images and cascade style sheets, we have the `static` folder. There are more folders for modules related to each repository and API accessed.

Inside each of these repository modules, we have the files related to the Django MVT architecture, like `models.py` and `views.py`. We do not have the template files because we use React for the frontend, but we have `urls.py` files for URL redirecting inside the modules. They hold a mapping between the endpoints called by the frontend and the views with the logic.

The `views.py` files deal with the backend logic. The `models.py` files describe the database schemas, with the tables as classes and the attributes inside each of them. Their relationships are also considered. We describe the database schema in its own section (section A).

We can bring the Django app up using the command

```
python manage.py runserver
```

The application will be available at `localhost:8000`

React

React was installed in the `frontend` folder. To install a React application, having the Node package manager installed, we simply have to enter:

```
npm -g install create-react-app  
create-react-app <app_name>
```

We can then start the application, navigating to the `app_name` folder and running

```
npm run start
```

By default, the application will be visible at `http://localhost:3000`

The node file that holds the package names and versions that must be present or installed for the React app to run the same way in every environment is `package.json`.

As modules are being added to the project throughout the implementation phase, the file is automatically updated.

react-router-dom⁶

We chose react-router-dom as our routing module to map the get requests into react components. It is installed with

```
npm install react-router-dom
```

material-ui⁷

Material-ui is an interface library for React that implements Google's Material Design. It has easy to incorporate, ready-made, clean, appealing to the eye components into any react application. It is installed with

```
npm install @material-ui/core
```

Redux

We use Redux to hold state between pages and specifically to deal with user authentication. Redux works with all types of javascript development, and there is a specific package for React: react-redux. To return functions from action creators in redux dispatch, we need to use a middleware called redux-thunk. We can install all in one go:

```
npm install redux react-redux redux-thunk
```

Axios⁸

Axios is a Javascript library that simplifies HTTP requests from node.js or XMLHttpRequests from the browser and supports promises, useful for asynchronous communication. It is installed with

```
npm install axios
```

Docker

Docker operates with containers and images. Each container wraps the code and all its dependencies for the correct use of part of an application. We have three containers and one image. The image is for the postgresql database, which just needs to be pulled, and then they turn into containers when ran. For the frontend, backend and Nginx service containers, we prepared a Dockerfile for each of them [15]. The Dockerfile calls commands defined by us to assemble and build an image to run the part of the application as intended. To run the whole application from a single command, we created a docker-compose.yml, a config file for docker-compose. It

⁶<https://reactrouter.com/>

⁷<https://material-ui.com/>

⁸<https://github.com/axios/axios>

configures and lifts at the same time multiple containers. Afterwards, we just need to run the `docker-compose up` command to deploy and access the whole application.

Also, environment (`.env`) files in the backend and postgres folders, postgres folder and project root folder were created. [15]

The final architecture of the project is the following:

```
DEDaMi/
  backend/
    django_app/ (*not showing the files inside the folders)
      converter/
      ensembl/
      genbank/
      go/
      files/
      kegg/
      mainapp/
      output/
      padel/
      pubchem/
      reqs/
      static/
      users/
      utils/
      manage.py
    Dockerfile
    entrypoint.sh
    .env
    requirements.txt
  frontend/
    react_app/ (*not showing the files inside the folders)
      node_modules/
      public/
      src/
      package.json
      package-lock.json
    Dockerfile
  nginx/
    Dockerfile
    nginx.conf
  postgres/
    .env
  docker-compose.yml
  Dockerfile
```

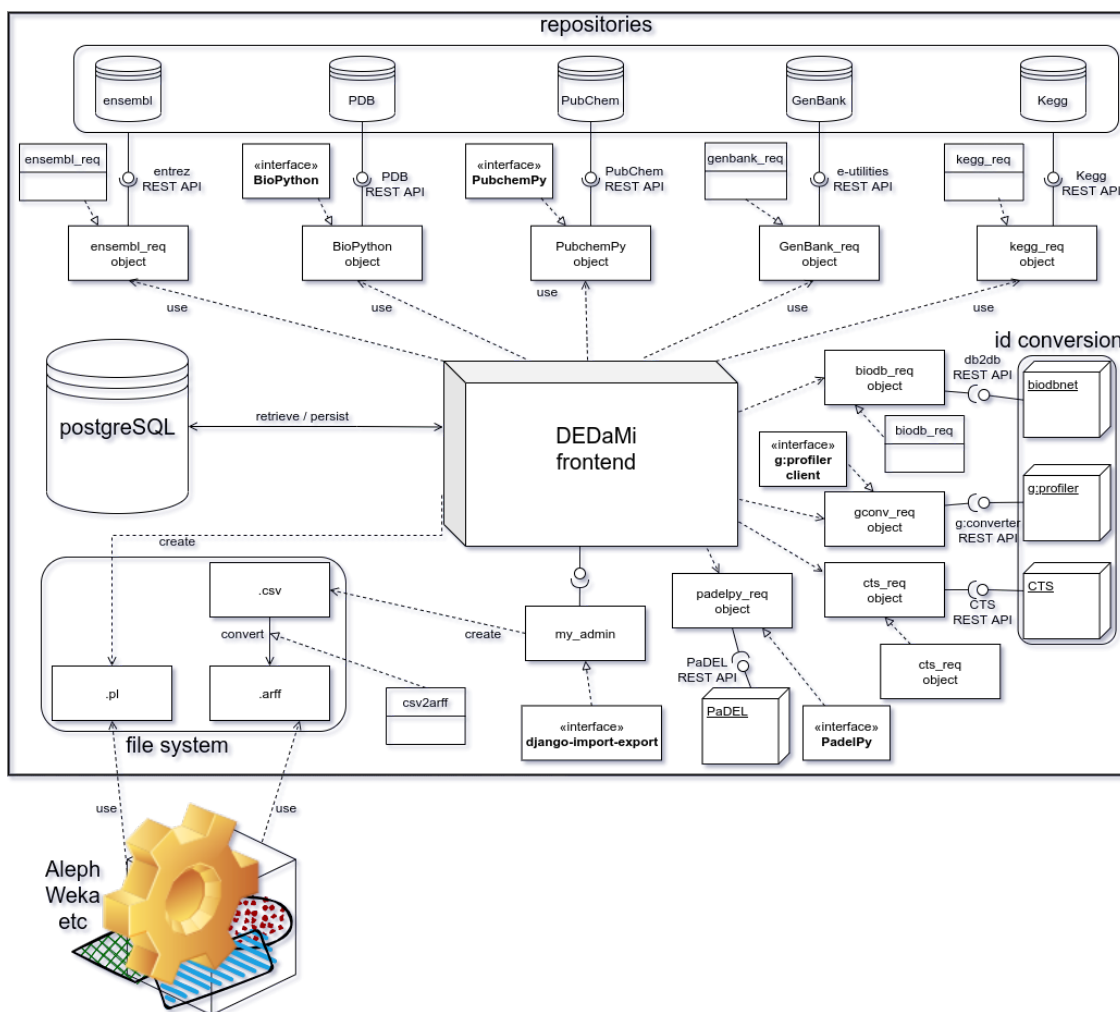


Figure 4.3: Application Architecture

4.4 Architecture

The DEDaMi application is split between the backend and the frontend. Frontend only deals with asking for input and choices and showing results and forms. The backend receives the user inputs and choices, uses them to ask the repositories and data-mining software for data, uses some wrappers, and returns to the front end to show it to the user. Also, in the backend, the database is accessed to save the data fetched from repositories and read it beforehand to access if the requested data by the user is already available locally. And for each record fetched conversions are made to other repository with more subsequent retrievals and generation and saving of Prolog clauses. Figure 4.3 illustrates a rough high-level architecture design of the platform. Besides the architecture, we will also discuss the different interfaces the user will deal with, the navigation flows, and possible user stories.

Frontend

Frontend endpoints

The endpoints available in the frontend are listed in Table 4.4.

Table 4.4: Frontend endpoints

Method	relative path	description
GET	/	Landing page
GET	/bulk	Bulk input of identifiers via file or textbox
GET	/output	List of output options
GET	/output/padel	Input of identifiers to get descriptors from
GET	/output/prolog	Input of identifiers to convert to Prolog facts
GET	/output/csv2arff	Input of a .csv file to convert to an .arff file
GET	/input	List of repositories available
GET	/genomics	List of Genomics repositories available
GET	/genomics/genbank	Genbank repository page
GET	/genomics/kegg	Kegg repository page
GET	/genomics/ensembl	Ensembl repository page
GET	/cheminformatics	List of Cheminformatics repositories available
GET	/cheminformatics/pubchem	Pubchem repository page
GET	/proteomics	List of proteomics repositories available
GET	/proteomics/pdb	PDB repository page
GET	/update_password	Page for the user to change password
GET	/admin	Administration page
GET	/login	Login page

Site Administration

Django automatically provides an administration section, where we can manage users and the databases. Previously we must create a superuser. This superuser will access the administration site `/admin`, and then he can add, remove, and manage other admins or users. To create a superuser, we just need to input the following command:

```
python manage.py createsuperuser
```

For the databases to appear on the administration page, we need to register the models in the `admin.py` file in each app folder.

We use the `django-import-export` library to handle the way databases and tables are shown, imported and exported.

django-import-export ⁹

With the help of this library, we can easily add the possibility of exporting tables from the database to several formats (csv, xls, and others) from the administration page. We install it with:

```
pip install django-import-export
```

In the end, an `admin.py` file would look like this stripped example from `pubchem`:

```
from django.contrib import admin
from import_export import resources
from import_export.admin import ImportExportModelAdmin
from .models import Compound

class CompoundResource(resources.ModelResource):
    class Meta:
        model = Compound
        exclude = ('created', 'modified',)

class CompoundAdmin(ImportExportModelAdmin):
    resource_class = CompoundResource

admin.site.register(Compound, CompoundAdmin)
```

But, because we created a customised class that can be called by every module so we wouldn't be repeating code again and again, this code ended slightly different in the `utils` package.

Frontend architecture

The react application entrypoint is the `Index.js` file, which points to the `App.js`. `App.js` calls the router in `Urls.js` and applies classes from the `react-router-dom`: `import {BrowserRouter, Route, Switch, Redirect} from "react-router-dom";`

Each route the `Urls.js` can call will use the master templates, in which the layout is filled with the requested page, and a `Repo` page is composed by blocks:

```
src>
  components>
    blocks>
      RepoForm
      RepoCheckGroup
      RepoRadioGroup
      TextInput
    comps>
      Breadcrumbs
      Help
      StyledLink
  master>
```

⁹<https://github.com/django-import-export>


```
    Layout
    TopBar
    Footer
  pages>
    Field
    Landing
    Repo
  images>
  setts>
  store>
```

In the `comps` subfolder, we have customizations of `material-ui` components: Bread-crumbs, Popover and Link.

The `images` folder hold several images, icons and logos used in the application. In the `store` folder, we find the `store`, `actions` and `reducers` for `Redux`. We use `Redux` in some cases to have the state available in all the layers of the component hierarchy at the same time.

We also use, in certain reusable modules, settings held in configuration files in the `setts` folder. The settings are then parsed by the application for the right appliance and mapping according to the page to display.

Landing Page

The `DEDaMi` application root points to a landing page where we have to choose between accessing the bulk input page, the output options page and the repositories page. The access is granted if authenticated. If we are not, we are redirected to the login page, and after authentication, we get redirected to our choice.

Repositories Page

In this page we can choose an available repository to access. Here, we choose which repository we want to make specific queries to.

Repository Access Pages

The pages where we choose which properties we want to fetch from the repositories. Here, we can choose the properties to access from the repository, the input we are supplying, and the output format we expect.

Bulk Input Page

To allow bulk inputs, either from a file or by inputting or pasting identifiers directly in a textbox, there is one special kind of Repository page that uses the same `React` construction logic, using the same `RepoForm`, `RepoCheckGroup`, `RepoRadioGroup` and `TextInput` components.

Outputs Page

In this page we choose between accessing one resource that produces Prolog facts from PDB ids and Gene ids or another resource that get PaDEL descriptors from Pubchem compound ids and saves Prolog facts using them.

PaDEL-Descriptor output Page

To ask the system to generate Prolog facts with chemical descriptors obtained from the PaDEL-Descriptor we insert Pubchem compound identifiers using this page. We can also choose which type of descriptors we are looking for: 1D/2D, 3D, fingerprints, all of these or one of the other three possible pair combinations.

Other Prolog outputs Page

We can ask the application to process PDB ids or Gene ids, either from a file or by inputting or pasting identifiers directly in a textbox, and save Prolog facts according to the available information about them.

Backend

Backend Endpoints

Axios handles the requests and responses to the backend. The parameters chosen by the user are sent to a function that directs the Axios request to the backend. Axios then send these values through a POST request and waits for the response. The backend receives the request with the parameters, and the appropriate function in the requested view processes the information. It verifies if the requested information is already available locally, in which case it will be returned to the frontend or if a call must be made to a repository.

Table 4.5: Backend endpoints

method	relative path	description
POST	/api/auth/logout/	Logout
POST	/api/auth/login/	Login
POST	/api/auth/update_password/	Change Password
POST	/api/admin/	Administration area
POST	/api/pubchem/	Pubchem repository logic request handler
POST	/api/kegg/	Kegg repository logic request handler
POST	/api/pdb/	PDB repository logic request handler
POST	/api/genbank/	Genbank repository logic request handler
POST	/api/ensembl/	Ensembl repository logic request handler
POST	/api/bulk/	Batch input logic request handler
POST	/api/padel/	Get PaDEL descriptors logic request handler
POST	/api/prolog/	Output to Prolog facts logic request handler
POST	/api/arff/	Convert a .csv file to an .arff file

Table 4.5 lists the available endpoint calls to the backend.

The request sent to the backend has the following format:

```
{
  "repo": <repository>,
  "inputvalue": <input value>,
  "input": <input supplied type>,
  "domain": [pubchem substance/compound],
  "output": [pubchem selected outputs],
  "outformat": [pubchem output format],
  "d3d": 2d/3d [pubchem compound],
  "file": [file uploaded]
}
```

The response from the backend has either the data with JSON with element or properties asked for or an error message and one of the following status codes:

- 200 : "OK"
- 422 : "Unprocessable Entity"

Backend architecture

In the backend, besides the normal Django architecture, we have a module for each of the repositories to access. There is also another module for the conversion tables logic and database and other useful separated folders.

```
django_app/
  converter/
  ensembl/
  files/
  genbank/
  go/
  kegg/
  mainapp/
  output/
  padel/
  pubchem/
  reqs/
  static/
  users/
  utils/
```

Each module folder has a migrations folder, with every change made in that module's models since creating the database schema. The files in the module folder include `models.py` with the table definitions, `urls.py` with module related sub-endpoints and `admin.py`. The `admin.py` file has templates and directives of what to be displayed in the administration area.

Most importantly, we have the `views.py` file with the logic for handling the front-end requests and creating an instance of an object of the repository class and calling its method(s). And in most of the modules there is also another package, named after

the module, that holds all the major logic, classes and methods regarding the module operations.

We may also have a `test.py` file with tests for the backend regarding that module and repository objects Repository modules

Repository modules

Each repository has its own module with its own database schema, a `views.py` file with the logic to handle the frontend requests, and a `<repo_name.py>` file with the class for the module's objects.

Converter module

The converter module holds the converter module with classes to handle the conversion of identifiers between different repositories and deal with the bulk requests. The folder also has the same structure and sub-folders as the repositories' folders.

The entry point sends the input request to a parser that identifies the type of identifiers introduced. The flow next retrieves information available locally if it exists or fetches from the appropriate repository if it does not. Using id converters it gets ids for the same entity on other repositories of the same field of study. Using this ids it will fetch the record if it is till not available locally. In the end we have a database entry that points to objects from each repository in the same field of study.

Output module

In this module we have the classes and methods to handle the requests to generate Prolog facts from PDB ids and Gene ids. It will try to obtain information in the local database and if there is still none it will access the repositories needed. It will then, from the attributes of the records, format the string to be saved into the Prolog knowledge base and to `.pl` files.

Padel module

This module deals with the requests to generate Prolog facts from Pubchem compound ids. It will try to obtain information in the local database and if there is still none it will fetch it from the Pubchem repository and save it locally and then ask PaDEL-Descriptor for descriptors. After saving new records to the database (not necessary of already has them), it formats the string to be saved into the Chemistry knowledge base and to `.pl` files.

Go module

Whenever a Go: id is saved in a database of a repository or conversion, its properties are fetched and a Go term and bioentity are also saved using the Go module.

The Go module consists of a simple class with a constructor that receives the Go: id and automatically fetches the properties from GOlr and BioLink endpoints. The Go class has only two methods, one saves the Term and the other all bioentities related to a Go term.

Testing

To run tests, we just have to input the following in the command line:

```
python manage.py test
```

Django automatically runs the unit test functions inside test classes on every `test.py` file in the project.

If we want to run tests for just one module, like Pubchem, for instance, we run:

```
python manage.py test pubchem
```

In addition, if we want a specific test, we include the name of the function of the intended test:

```
python manage.py test pubchem.test.PubchemTestCase.test_substance_by_sid
```

Django unit tests usually inherit the `TestCase` class but, because we launch separate threads and `TestCase` runs inside of a transaction we could not access the conversion tables from a repository test. To achieve the intended goal of the tests we need to use `TransactionTestCase`, so that the test process runs in the regular auto-commit mode.

Cross-module utility functions

The `utils` folder consists of only one file (`utils.py`) with methods used by any class in the application and also constants like regex patterns and others.

Repository requests classes

The `reqs` folder includes packages with classes to make and handle requests to the repositories and conversion tools. These include `Biodb`, `CTS`, `Kegg`, `Ensembl`, `Entrez` and `G:Profiler`.

4.5 User Stories

In Tables 4.6, 4.7 and 4.8, we enumerate the main user stories of the application and how to achieve them. The main possible uses of the application are listed in the table. There can be additional, more specific user stories for different repositories, like searches constrained to a particular database. Searches are not available on some repositories, and on others, we can specify which properties we want to be returned.

When a user specifies an identifier in a `Repository` page, the application will search in the local database and only if the record is not found it will access the repository to get it and save it. The properties of the object pointed by the provided identifier are returned to the frontend. In the background, a new thread is launched to convert the identifier to other repositories of the same field of study, get the object from converted identifiers in the other repositories, and relate them in a dictionary-like table.

Table 4.6: Main User stories, their usages and corresponding UI Figures

User Story	Usage	UI
Export Database	Go to the administration area by clicking the symbol and there click export in the chosen table	Figure B.25
Insert identifiers in bulk in a textbox	Access the Bulk Page via Landing Page and type or paste identifiers in the textbox	Figure 4.8
Provide identifiers in bulk from a text file	Access the Bulk Page via Landing Page and click the select file button	Figure B.3
Get chemical descriptors from PaDEL (and choose which) and generate Prolog clauses	Access the PaDEL page via Output Page and paste/write Pubchem compound ids or click the select file button	Figure 4.9
Generate Prolog clauses from protein or gene ids	Access the Prolog page via Output Page and paste/write PDB ids or gene ids or click the select file button	Figure B.6
Provide a(n exported) .csv file to convert to an .arff file	Access the Csv2arff Page via Output Page and click the select file button	Figure B.5
Search for terms in some repositories and get a list of identifiers matching that query	Input a query on a repository page	Table 4.7
Save objects properties (in each repository ^a)	Provide identifiers in a Repository page or in the Bulk Input Page	Table 4.7
Convert identifiers between repositories	Provide identifiers in a Repository page or in the Bulk Input Page	Table 4.7
Keep a translation table between different repositories (for each field ^b)	Provide identifiers in a Repository page or in the Bulk Input Page	Table 4.7
Query each repository and see the properties for the identifier asked for	Specify an identifier on a repository page	Table 4.7
Authenticate Change password Add user Edit user		Table 4.8

^aPubchem, PDB, Kegg, Ensembl, Genbank^bCheminformatics, Proteomics, Genomics

Table 4.7: Repositories corresponding UI Figures

Repository input page	UI
Pubchem Repository page	Figure B.10 Figure B.11 Figure B.12
PDB Repository page	Figure B.7
Kegg Repository page	Figure 4.7
Genbank Repository page	Figure B.15
Ensembl Repository page	Figure B.19

If the user takes advantage of the bulk input feature, the identifiers will be first parsed to identify where to look locally and the correct repository to access, and then the flow takes the same course as the one described for the Repository use case described above. Nonetheless, there is one major difference because all the bulk process is handled quietly in the background by new threads. Therefore, and because, presumably, more than one identifier was provided, no properties are shown in the frontend. Instead, a message will be shown saying that when the background process is over, a message will be sent to the user's stored email account.

Table 4.8: User management User stories, their usages and corresponding UI Figures

User Story	Usage	UI
Authenticate	Go to the Login Page either by clicking the symbol or trying to access a protected resource	Figure B.23
Change Password	Click the symbol on the topbar	Figure B.24
Add user (administrator)	Go to the administration area by clicking the symbol and there click on Add User (and give privileges)	Figure B.26
Edit user	Go to the administration area by clicking the symbol and there click on the User and edit the information	Figure B.27

4.6 User Interface

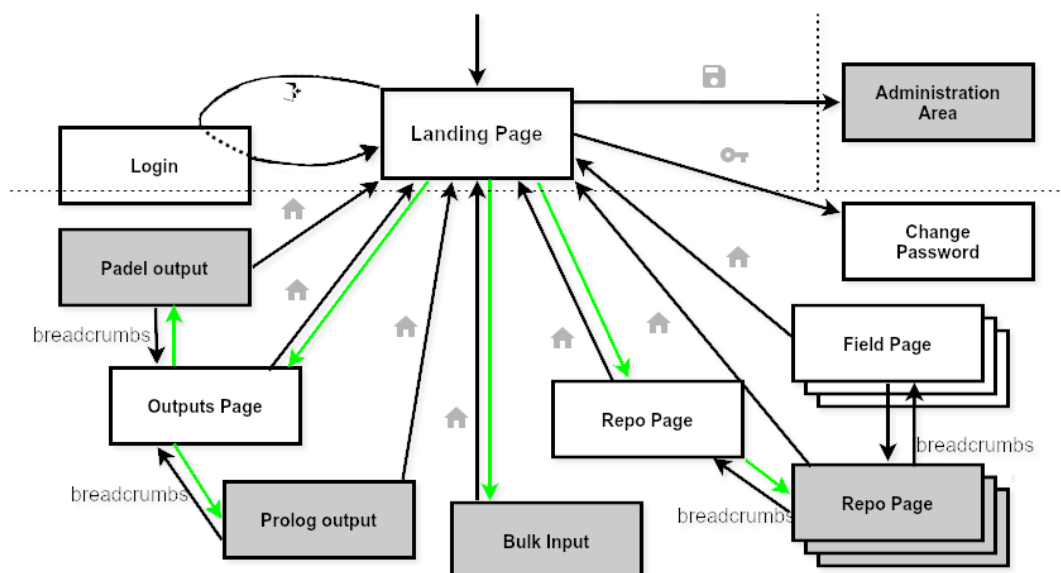


Figure 4.4: Navigation Flow

For a first glance on the whole frontend navigation flow, we can observe Figure 4.4. The first page displayed when accessing the application is the Landing Page (Figure 4.6). Authentication is mandatory in order to access any other resource.

A user logs into the platform if he clicks on the Login symbol in the top bar or as soon as he tries to access a protected resource. Every page is a protected resource apart from the Landing Page.

At the Landing Page an user must chose between three areas: Bulk Input; PaDEL / Prolog outputs; and Repositories. Apart from the Administration Area, we can access the Landing Page by clicking the home button in the top bar (4.6) from anywhere in the application.

The main pages for a user to visit are the Repository Pages, the Bulk Input page, the Output pages and the Administration Area, where database export is possible.

In the Output Page, the user chooses between sending Chemical data to PaDEL to get descriptors or to convert protein or gene ids to Prolog facts.

In the Repositories Page, the user chooses which repository he wants to access for specific searches.

Interface

Here we present the pages displayed to the user, allowing data input and showing the responses from the server.

More examples of interface page can be viewed on appendix B

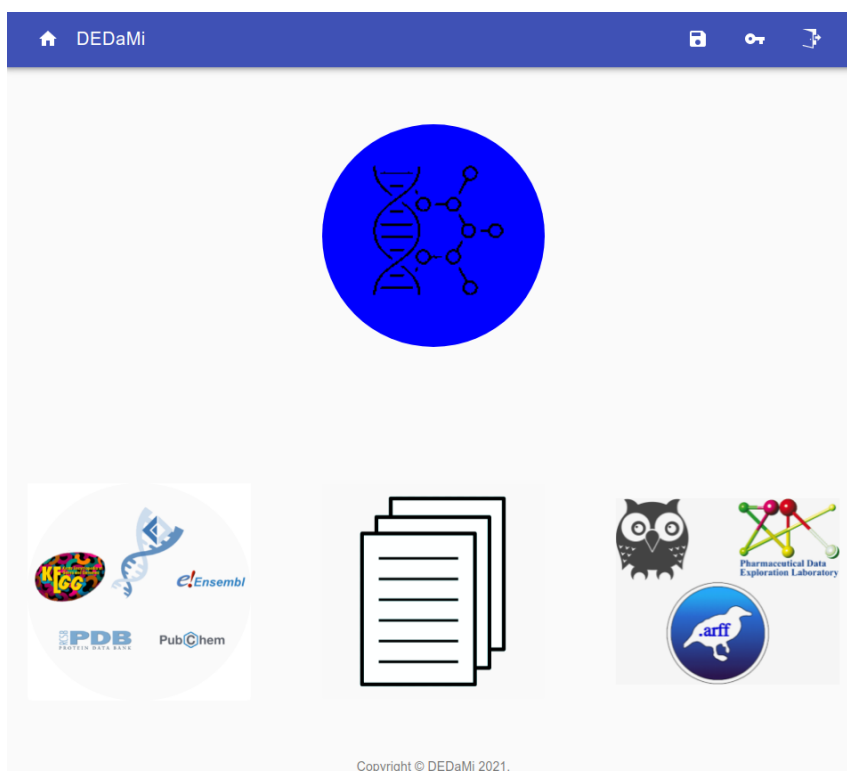


Figure 4.5: Landing Page

Landing page

On the landing page, all that a guest or registered user can see are the avatars of the highest level pages: Bulk Inputs Page; Outputs Page; and Repositories Page.

Topbar

The top bar has a direct link to the landing page on the left side. On the right side, if we are authenticated, we can find three links. At the far right, we have the logout (login if we are authenticated), then the link for changing our password and finally a link to access the administration area, or export database area, with a floppy disk symbol. Not in the top bar, but directly below it, we have the breadcrumbs. This component helps us navigate through the application and helps us situate in the application.

Field page

The field page is similar to the Landing Page, restricted to the repositories related to the given field of study, and only accessible by authenticated users (Figure B.1). The only way we can navigate to a field page is through the breadcrumbs on a Repository Page.

Repositories page

The user is presented with the avatars (working as buttons) representing each repository Page to choose which repository to access (Figure 4.6).



Figure 4.6: Repositories page

Repository Access pages

For each repository that the application uses, a specific page is displayed. This way, the user can interact just with that repository and with repo-specific actions.

We will list what can be done on each repository page.

Pubchem

On the Pubchem page, which is the one with the most options, we can choose between the substance and the compound modes, using a slider on the top. If we choose the compound mode, we have another slider that we can use to specify if 3D properties should be returned or just 2D. Because of how the pubchempy wrapper works, accessing different information without combining it, it is better to work like this (Figure B.13).

In the compound mode, we can input the pubchem compound identifier (CID), one of its names, the formula, the inchi identifier, the inchikey hash or the smiles notation. The possible inputs in the substance mode are only the Pubchem substance identifier (SID) and one of the possible names.

Pubchem makes a distinction between compounds and substances. A compound is a standardized chemical structure representation present in possibly more than one substance.

On the Pubchem page, we can choose which properties to be presented in the interface. There are only four properties possible for a substance: Synonyms, Stan-

standardized Compound, CID and AID. A compound has a larger number of possibilities¹⁰, 27 to be exact, to which we can still add eleven exclusive to 3D¹¹. Of course, we can simply check the "all" option.

Regardless of which properties we choose to see, all possible properties will be saved in the database.

PDB

On the PDB page, we only have two input possibilities. We can specify a PDB identifier that already known to us to see its properties and save them to the database or enter a query to get identifiers related to the term queried (Figure B.7, B.8 and B.9).

Kegg

On the Kegg page, the input possibilities are divided between searches for terms in the Kegg databases and inputs of known identifiers to see its properties and save them to the database (Specify Entry). If we want to find identifiers that match a term, we can search in a specific Kegg database¹² or on all of them at once. The search results will be presented on the same page, and we can then use one of the identifiers in the Specify Entry input box.

We can see the Kegg input page in Figure 4.7 and search results in Figure B.14.

Genbank

On the Genbank page, we only have two input possibilities. We can specify an identifier (gene id, accession or gene name) that we already know to see its properties and save them to the database or enter a query to get identifiers related to the term queried (Figure B.17 and B.18).

Ensembl

On the Ensembl page, we can only specify an Ensembl identifier that we already know. The application will show the properties of that identifier and save it (and also identifiers on other related repositories) to the database (Figure B.20, B.21 and B.22).

Bulk Inputs Page

To introduce bulk requests of several identifiers, the user must access the bulk page at /input/bulk. On the Bulk page, the user has two options: uploading a file with the identifiers or writing/pasting into a textbox (Figure 4.8).

Both alternatives follow the same format of one instruction or identifier per line. The preferred mode of input is through a perfectly unique identifier, but sometimes that is not possible, namely when introducing gene ids.

¹⁰Molecular Formula, Molecular Weight, Canonical SMILES, InChI, InChIKey, Isomeric SMILES, IUPAC Name, Synonyms, XLogP, Exact Mass, Monoisotopic Mass, TPSA, Complexity, Charge, HBond Donor Count, Rotatable Bond Count, Heavy Atom Count, Isotope Atom Count, Atom Stereo Count, Defined Atom Stereo Count, Undefined Atom Stereo Count, Bond Stereo Count, Defined Bond Stereo Count, Undefined Bond Stereo Count, Covalent Unit Count, SID, AID

¹¹Volume 3d, Multipoles 3d, Conformer RMSD 3D, Pharmacophore Features 3d, MMFF94 Partial Charges 3d, MMFF94 Energy 3d, Conformer ID 3d, Shape Selfoverlap 3d, Feature Selfoverlap 3d, Shape Fingerprint 3d and Effective Rotor Count 3D

¹²Search Pathway, Search Gene, Search Disease, Search Enzyme, Search Drug, Search Compound, Search Reaction, Search Orthology

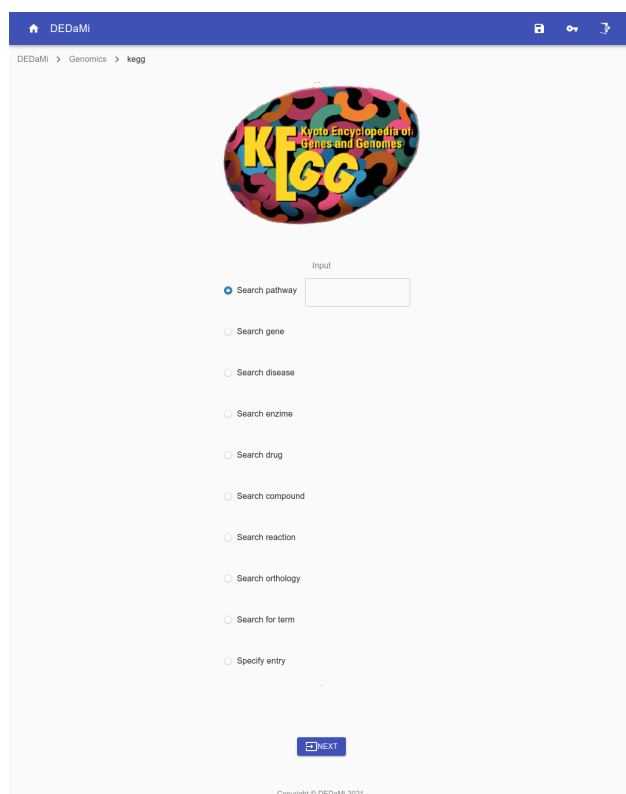


Figure 4.7: Kegg input page

For those disambiguation occasions, we have introduced two higher optional levels. We call these levels header and field. The header is the level directly above the identifier and tells the application to which repository send the identifiers on the lines directly below it. The field level identifies the field of study of the identifiers following it, 'chem', 'gen' or 'prot'.

Whether being uniquely identifiable identifiers or even properties (inchikey and more) or ambiguous or strange strings, the system, with the help of the headers and fields, if present, will try to correctly identify the input and save the records directly asked, those related to it and the conversion table.

All these processes are made in the background, using threading. When the task completes, the user will receive an email message at the email associated with his account, telling that the job has finished and with information about the number of inputs processed and conversion records saved.

The following list is a possible input (without the comments, which are only present to describe the identifier on each line)

```
cid5090 # Pubchem cid with 'cid' prefix
SID6433516 # Pubchem cid with 'SID' prefix
KEGG10458 # Gene number with 'KEGG' prefix
D10453 # Kegg drug id
c21200 #Kegg compound id
NM_001206992 # RefSeq transcript accession
NP_001193921 # RefSeq protein accession
NG_029843 # RefSeq gene accession
```

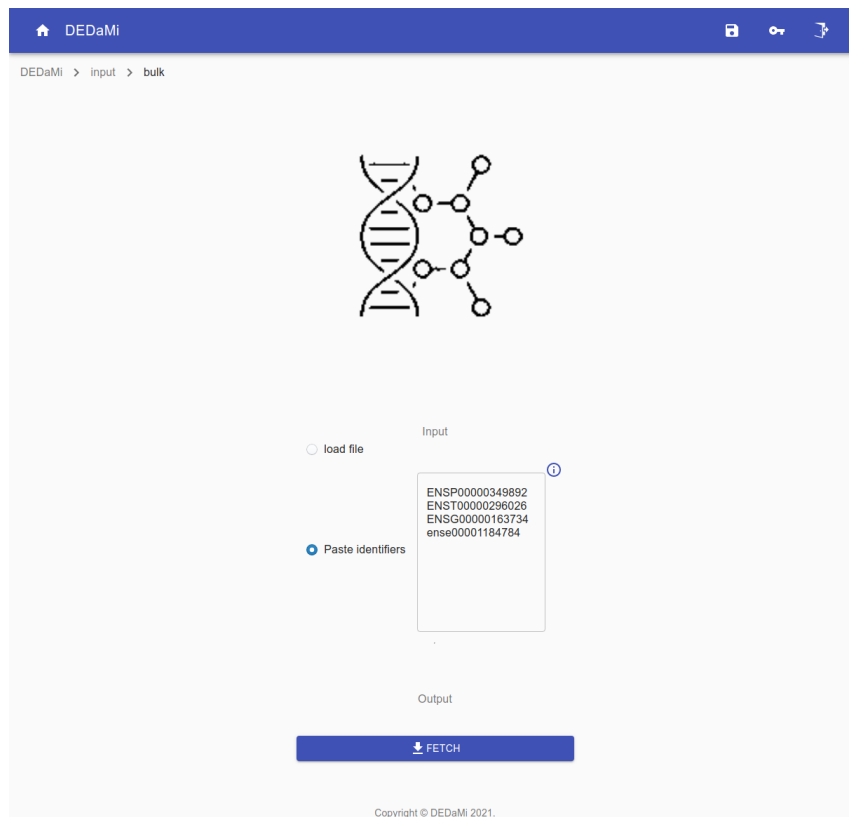


Figure 4.8: Bulk input page

```

ENSP00000349892 # Ensembl protein id
ENST00000296026 # Ensembl transcript id
ENSG00000163734 # Ensembl gene id
ense00001184784 # Ensembl exon id
BQJCRHHNABKAKU-KBQPJGBKSA-N # InChiKey
Nr1H4 # gene symbol (will try to identify)
PDB5a39 # pdb_id with 'PDB' prefix
6XL5 # pdb_id (will try to identify)

```

If we want to pass ambiguous identifiers using headers:

```

genbank
NR1h4
10458

pdb
5a39
6XL5

```

A blank line can be used to force the reset of the settings of the previous block. If after a blank line comes ambiguous identifiers, the application will not know the repository and can only try to identify.

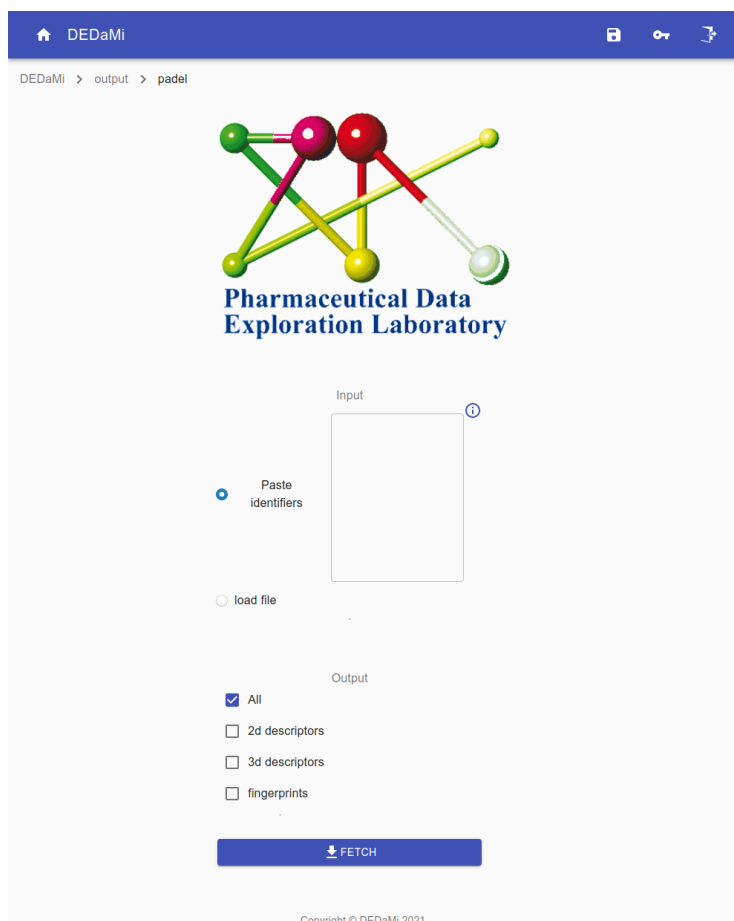


Figure 4.9: PaDEL Page

Outputs page

In this page, the user must choose between PaDEL or PDB/gene Prolog pages (Figure B.4).

PaDEL page

Page in which the user tells the system which compound ids he wants to get PaDEL descriptors from and generate Prolog clauses. The ids can be introduced in a textbox or selecting a file and he can choose which descriptors (Figure 4.9).

Prolog page

Page in which the user tells the system which PDB ids or gene ids he wants to generate Prolog clauses with. The ids can be introduced in a textbox or selecting a file and he can choose which descriptors.

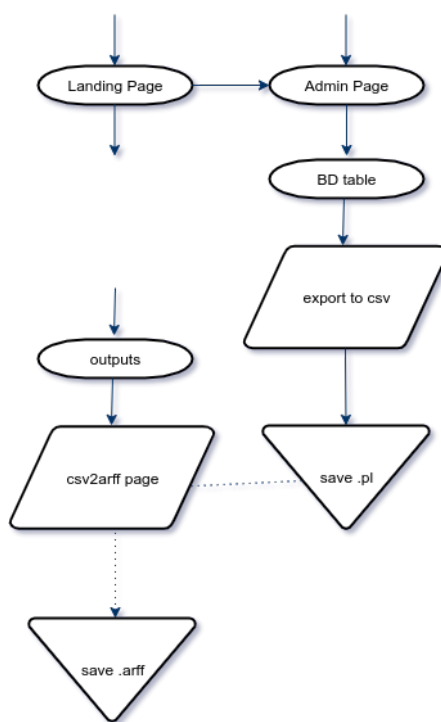


Figure 4.10: .csv to .arff pipeline

Csv2arff page

Page in which the user provides the system with a .csv file to be converted to a Weka .arff file. The backend will save the converted file in the files/arff/ folder in a file with the format <yyyyMMddhhmmss>.arff. After conversion, the interface will show the user the name of the converted file. The full flow of the .csv to .arff can be seen in Figure 4.10.

Django administration WELCOME, ADMIN VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Converter > Chemistrys

Select chemistry to change IMPORT EXPORT ADD CHEMISTRY +

Action: Go 0 of 2 selected

<input type="checkbox"/>	INCHIKEY	PUBCHEM COMPOUND	PUBCHEM SUBSTANCE	KEGG C	KEGG D	CHEBI	CHEMBL
<input type="checkbox"/>	XCVBYOXRSFQBH-AXDSSHIGSA-N	Compound object (119)	Substance object (97)	Compound object (93)	-	-	-
<input type="checkbox"/>	RZJQNCSTQAWON-UHFFFAOYSA-N	Compound object (122)	Substance object (99)	-	-	-	-

2 chemistrys

Figure 4.11: Cheminformatics conversion table in Administration Area

Administration Area

The administration area is the regular Django administration area but with templates to show attribute tables and import/export possibilities provided by the `django-import-export` library. In Figure 4.11 we see an example of database table view inside the Administration Area.

Only an administrator can add and edit user accounts, but we can give other user permissions to access the database and export the tables. In the scope of this work, we can create a user group with the right permissions and add users to that group.

4.7 Requests & Repositories

We access some of the repositories and data-mining calculators either through GET requests to their REST API endpoints or using open-source freeware wrappers and libraries written specifically for that effect.

Pubchem & PubChemPy

PubChemPy is used to ease the access, requests and parsing of the responses to and from the Pubchem repository. It is installed with

```
pip install pubchempy
```

The main methods used with this library are the following:

- `get_compounds(value, property)`
- `Compound.from_cid(cid_value)`
- `get_substances(value, property)`
- `Substance.from_sid(sid_value)`
- `get_cids(value, property, [...])`
- `get_sids(value, property, [...])`

We can search for substances or compounds. If we are after compounds, we have some properties only available if we choose the 3D mode. We can obtain all properties at once or just some. The available properties are shown in Table 4.9.

PaDEL-Descriptor & PaDELPy

PaDELPy allows direct access to the PaDEL-Descriptor command-line interface via Python. It is installed with

```
pip install padelpy
```

The main methods used are:

- `from_smiles(smiles_list)` - gets a list with the descriptors for the molecules with smiles supplied in the list. We can also provide further parameters to indicate which type of descriptors we want and timeout instructions, for instance.

Table 4.9: Properties obtainable from Pubchem

Domain	Property	obs.
Compound	cid	
Compound	canonical_smiles	
Compound	charge	
Compound	complexity	
Compound	exact_mass	
Compound	isomeric_smiles	
Compound	iupac_name	
Compound	molecular_formula	
Compound	molecular_weight	
Compound	monoisotopic_mass	
Compound	tpsa	
Compound	xlogp	
Compound	atom_stereo_count	
Compound	bond_stereo_count	
Compound	covalent_unit_count	
Compound	defined_atom_stereo_count	
Compound	defined_bond_stereo_count	
Compound	h_bond_acceptor_count	
Compound	h_bond_donor_count	
Compound	heavy_atom_count	
Compound	isotope_atom_count	
Compound	rotatable_bond_count	
Compound	undefined_atom_stereo_count	
Compound	undefined_bond_stereo_count	
Compound	sids	
Compound	aids	
Compound	synonyms	
Compound	volume_3d	only in 3d
Compound	conformer_id_3d	only in 3d
Compound	conformer_rmsd_3d	only in 3d
Compound	multipoles_3d	only in 3d
Compound	pharmacophore_features_3d	only in 3d
Compound	mmff94_partial_charges_3d	only in 3d
Compound	mmff94_energy_3d	only in 3d
Compound	shape_selfoverlap_3d	only in 3d
Compound	effective_rotor_count_3d	only in 3d
Compound	feature_selfoverlap_3d	only in 3d
Compound	shape_fingerprint_3d	only in 3d
Substance	sid	
Substance	cids	
Substance	aids	
Substance	deposited_compound	
Substance	standardized_compound	
Substance	synonyms	

- `from_md1(md1_file)` - gets a list with the descriptors for the molecule in the supplied md1 file. We can also provide further parameters to indicate which type of descriptors we want and timeout instructions, for instance.
- `padeldescriptor(list_with_property:value_pairs)` - sends a set of properties and settings to configure the PaDEL descriptor options.

PDB & BioPython

To access the PDB REST API we chose to use Biopython. Biopython is a suite of modules to access several bioinformatics resources. We describe the ones we use.

To use the Biopython parser we first need to download a PDB/mmCIF file for a specific protein or aminoacid. Then we use the Biopython `pdb/mmCIF` parser module to parse the file and extract the properties and information we want.

Main methods used:

```
file = PDBList().retrieve_pdb_file(query.lower(), pdir='filepath')
```

and a mmCIF file is saved on the supplied directory with the data corresponding to the structure with the PDB identifier in the query.

```
dicto = MMCIF2Dict.MMCIF2Dict(file)
```

and a dictionary with all the data available in the file specified is created, from which we can obtain properties directly like:

```
complete_sequence = dicto['_pdbx_poly_seq_scheme.mon_id']
```

or through a transposition, using the python `zip()` method, like it is done in the method to get all the sheets with their properties (early short version):

```
sheets = [
    dict(
        zip(
            ('sheet_id', 'sheet_type', 'number_strands', 'details'),
            col
        )
    )
    for col in zip(
        dicto['_struct_sheet.id'],
        dicto['_struct_sheet.type'],
        dicto['_struct_sheet.number_strands'],
        dicto['_struct_sheet.details']
    )
]
```

We can also get counts directly from the dictionary

```
number_of_helices = len(dicto['_struct_conf.id'])
number_of_sheets = len(dicto['_struct_sheet.id'])
```

Table 4.10: Kegg entity types, examples and regular expressions

Entity	examples	regex
orthology	K00161, K00162, K00163, K00627, K00382	K\d{5}
compound	C00031	C\d{5}
disease	H00004	H\d{5}
drug	D01441	D\d{5}
drug group	DG00710	DG\d{5}
glycan	G00109	G\d{5}
module	M00010	M\d{5}
reaction	R00259	R\d{5}
reaction class	RC00046	RC\d{5}
network	N00002	N\d{5}
network variation map	nt06210	nt\d{5}
genome	T01001 (hsa)	T\d{5}\s\(.*\)
enzyme	ec:2.7.10.1	(?i)EC:\d{1,2}\(\.\d{1,2}){3}
gene	hsa:3643, vg:155971, vp:155971-1, ag:CAA76703	(<org> ^a vg vp ag):.*
pathway	map00010, hsa04930	(map ko ec rn <org> ^a)\d{5}
brite	br:08303, br:01002, br08303, ko01002, jp01023	-
variant	hsa_var:25v1	-

^athree or four-letter character substring that match any element of the list of Kegg organism codes

Kegg

The Kegg requests are provided by the `kegg_request` method that can be found on the `reqs.kegg_rest` package:

```
## Kegg REST API base url
request_url = 'http://rest.kegg.jp'

def kegg_request(request_type, request_value):
    if request_value.upper().startswith('NT'):
        request_value = 'ne:' + request_value
    url = f"{request_url}/{request_type}/{request_value}"
    return requests.get(url, timeout=(60, 60))
```

We can ask Kegg for a list of all pathways from a specific organism or just of the ones that match a query (e.g. "hsa,corona" for all human pathways with corona in the description). The main uses are these:

```

find = kegg_request('find', f'{search_type}/{query}')
response = kegg_request('get', newinput)

def get_find(self, term, database):
    return kegg_request('find', f'{database}/{term}')

```

The entities we can extract from Kegg are listed in Table 4.10 along with examples of codes and regular expressions to identify which entity the code refers to.

The attributes we extract and save from each of these entities are displayed in the database schema (Figure A.3).

GenBank

GenBank returns an XML response that can be easily parsed, despite the abuse of Gene-commentary tags, and similar ones, and without any verticality.

We use two of the possible endpoints from eutils. The `esummary` and the `efetch`. The `esummary` has fewer details but readily accessible at a higher level.

To allow a higher request frequency to the repository, we have created an application key that we can use in our requests by registering our app in the NCBI REST API. NCBI's eutils requests are not database dependent, so we indicate the database in the request URL:

```

response=requests.get('https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=
gene&id={id}&retmode=xml&api_key={NCBI_KEY}')

```

To process the xml we use the `ElementTree(etree)` API from the `lxml` XML toolkit:

```

tree = etree.fromstring(response.text) # or response.content

```

To parse the xml we use the `xpath` rules with the `ElementTree.xpath()` method. Example to get the `ensembl` id from a GenBank `efetch` record using `xpath`:

```

ensembl_id = tree.xpath("string(//Dbtag_db[text()='Ensembl']/../Object-id/*/text())")

```

Ensembl

We have four different entity types: genes (ENSG prefix), transcripts (ENST), proteins or translations (ENSP) and exons (ENSE). One gene may have several transcripts, and each transcript several exons and translations. Whatever entity is specified, the application will fetch and save information for the whole tree.

The Ensembl requests are provided by the `EnsemblRequest` class.

We use the following REST endpoints:

- `https://rest.ensembl.org/lookup/id/<gene_id>?content-type=application/json;expand=1`
to fetch a summary of the gene properties, one of which is the gene symbol, necessary for the next endpoint.
- `https://rest.ensembl.org/lookup/symbol/homo_sapiens/<gene_symbol>?content-type=application/json;expand=1`

to fetch the complete gene record, with transcripts, exons and proteins (but without the sequence).

- https://rest.ensembl.org/sequence/id/<gene_id>?content-type=text/plain

to fetch the gene sequence.

Go

To get the properties from a Go: id (Gene Ontology identifier) we access the Gene Ontology Resource REST API¹³.

To get all meta data about the GO term we use the GO Solr search engine API called GOLr^{14,15}.

Example of a GOLr endpoint:

```
http://golr-aux.geneontology.io/solr/select?fq=document_category:"ontology_class"&q
=**:&fq=id:"GO:0030182"&wt=json
```

To get the relationships between Go terms we use the BioLink Data Model^{16,17}, in which a GO term is referred to as function.

Example of a BioLink endpoint:

```
http://api.geneontology.org/api/bioentity/function/GO:0006915
```

bioDBnet db2db

The default id converter for genes and proteins is bioDBnet's db2db. It is accessed using creating a Biodb instance and getting its converted attribute. This class is located in the biodb.py package inside the reqs module.

In the `__init__()` method we have:

```
self.converted = requests.get(self.biodb_url % (
    source.strip(' ').lower(), value, target), timeout=(30, 30))
```

CTS

The Chemical Translation service is the default converter used for chemistry id conversions. Once again we just need to create an instance of the Cts class located in the cts.py package inside the reqs module.

The converter attribute points to the response, inside `__init__()` method of:

```
requests.get(f"self.cts_url/self.source/self.target/self.value",
    timeout=(30, 30))
```

G:profiler

When db2db is unable to convert the ids, we can try G:profiler's g:Convert for a second try.

To access g:profiler's g:convert, we use its official Python client gprofiler-official.

¹³<http://geneontology.org/docs/tools-guide/>

¹⁴<http://wiki.geneontology.org/index.php/GOLr>

¹⁵<https://github.com/geneontology/amigo/blob/master/golr/solr/conf/schema.xml>

¹⁶<https://github.com/biolink/biolink-model>

¹⁷<https://github.com/biolink/biolink-model/blob/master/json-schema/biolink-model.json>

In the `gpconvert.py` package inside the `reqs` module, we have the `__init__()` method that automatically does the request:

```
gp.convert(organism='hsapiens', query=[self.value],
           target_namespace=self.target)
```

To easily obtain a gene id from gene name we use this method that calls `g:profiler's` `g:convert:`

```
def name_to_uid(self, name):
    """converts a gene name/symbol into a genbank uid"""
    g_prof = GProfiler(return_dataframe=False)
    try:
        return g_prof.convert(organism='hsapiens', query=[name], target_namespace='
        ENTREZGENE_ACC')[0]['converted']
    except:
        return 0
```

Conversion

Each field of study has a table that relates the tables from each repository, namely Cheminformatics, Genomics and Proteomics (Figure A.7). The Cheminformatics translation table relates compounds and drugs obtained with pubchem and Kegg, as well as their ids in ChEmbl and Chebi. The Genomics table relates gene records obtained from Kegg with Genbank and Ensembl. The Proteomics table deals with proteins from Ensembl, Genbank, Kegg (enzymes), and also Uniprot ids.

Whenever an entry from one of the repositories is saved, all the other related are also accessed. The conversion tables relating to all of them are updated. All made in the background, not delaying the response to the frontend because a new thread is launched to handle the conversion and saving in other repositories.

Example of a thread creation after a compound is requested in pubchem, calling the Converter class to access the other repositories with compounds:

```
from converter.views import Converter
import threading

def foo(bar):
    ...
    t = threading.Thread(target=call_converter, args=['chem', 'Pubchem CID', pubchem.
    entry, pubchem.id_, pubchem.shared], daemon=True)
    t.start()
    ...

def call_converter(conversion, repo, entry, id_, shared):
    Converter(conversion=conversion, repo=repo, entry=entry, id_=id_, shared=shared)
```

We use different online conversion tools, according to the field of study.

For cheminformatics, we use the `cts` converter to get the id of the same compound in another database, using the InChi Key as starting point. As we do not have an inchikey in a Kegg compound record, we thought a suitable workaround was to use the pubchem and chebi properties in it to convert to inchikey.

The biodb db2db is used in genomics conversion. In this case, the Entrez Gene ID is used to convert from one database to the others. We also use g:profiler but just to easily obtain a Genbank gene uid from its name.

In Proteomics, the conversion is more difficult as one entry in one database may relate to many on another. We chose to use the Ensembl Protein Id as it offered a one-to-one conversion between Genbank, Uniprot and Ensembl. The biodb db2db is also used in the proteomics conversion.

These conversions are made using instances of objects of the class representing each conversion tool.

4.8 Deployment

The development of this application included the dockerization in four containers: Frontend, Backend, Database and nginx. Wrapping the containers, a docker-compose lifts all of them at the same time.

To install Docker Engine¹⁸ in Linux:

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl software-properties-
common
curl -fsSL https://download.docker.com/Linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/Linux/ubuntu $(
lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

To check if it is correctly installed:

```
docker run hello-world
```

To install docker-compose¹⁹ in Linux:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose
-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

To verify that it was correctly installed:

```
docker-compose --version
```

The Docker image is pushed to Docker Hub.

```
Docker hub account name: up201607946
App name: dedami
```

To connect to the Docker hub from the terminal, first, we need to login:

```
docker login
```

To build the image, in the binaries root folder, run the following command:

¹⁸<https://docs.docker.com/get-docker>

¹⁹<https://docs.docker.com/compose/install/>

```
docker build -t up201607946/dedami .
```

The next command is used to push the image to Docker Hub. We can add a tag for future references.

```
docker push up201607946/dedami[:tag]
```

To pull the image from Docker Hub, we just need to change push to pull:

```
docker pull up201607946/dedami[:tag]
```

We may want to save the image (to export):

```
docker save --output <filename>.tar up201607946/dedami
```

In that case, we use this command to load the image:

```
docker load --input <filename>.tar
```

If docker-compose is installed and we have either the binaries or an image built, saved or pulled, to deploy we just need the following command:

```
docker-compose up
```

If the image for a container is not found, but the binaries are there, docker-compose will build a new image before lifting all the containers.

Deployment in a DigitalOcean²⁰ Docker Droplet

We chose to deploy this project in a DigitalOcean droplet because it is very easy and simple. Since we already have the Docker containers and the docker-compose.yml, we just need a Linux server that either already has docker-compose installed or allows us to install it.

We access a DigitalOcean droplet using a remote connection with ssh. Inside the droplet, we just need to pull the image from Docker or the binaries from a git repository. Then lift with the docker-compose, adding the option to leave it running in the background:

```
docker-compose up -d
```

In addition to the Docker droplet, we need a physical volume in DigitalOcean, and we need to mount it.

To allow the database to be persistent between deployments, we need to have a volume for the database in docker-compose and to create a symbolic link from that volume name pointing to the volume mount.

²⁰<https://www.digitalocean.com/>

4.9 Summary

We have covered and detailed the problem we intend to solve, the basis of the solution we propose, in terms of technologies, case studies and approaches, and ended describing the implementation and final presentation of this specific solution.

Chapter 5

Case Studies

In this chapter we describe how we have processed the introduction of ids and retrieval of Prolog files with clauses for the ILP.

5.1 Handling Proteins

We chose to download pre-compiled culled PDB lists from PISCES: A Protein Sequence Culling Server¹.

The chosen list has the following constraints:

- Maximum pairwise per cent sequence identity : 95.0
- Resolution (X-ray and EM) : 0.0 - 5.0
- Do not include chains with chain breaks
- Chain length : 40-10000
- Maximum R-value (X-ray only) : 1.0

This file comprises 45163 lines that correspond to PDBchains but have fewer PDB structure IDs. The data is presented in six columns, separated by tabs, as we can see in the following example:

PDBchain	len	method	resol	rfac	freerfac
5D8VA	83	XRAY	0.48	0.072	0.078
3NIRA	46	XRAY	0.48	0.127	NA
5NW3A	54	XRAY	0.59	0.135	0.146
1UCSA	64	XRAY	0.62	0.139	0.155
3X2MA	180	XRAY	0.64	0.122	0.129

A specific Python script was written to parse the PDB ID to write the first four characters of each line (PDB ID from PDB chain) to a new file. This file was then selected in the Bulk Inputs page and the PDB tables populated.

¹<http://dunbrack.fccc.edu/pisces/>

Proteomic Knowledge Base

As the clauses concerning PDB structures are only generated from PDB id inputs, we decided that when there is any input of any PDB id, automatically the system creates facts and adds them to the Proteomic knowledge base. Nonetheless, a user can still introduce PDB structure ids in the Output Prolog page to generate facts. This page is prepared to receive either Gene ids or PDB ids. Every time a PDB structure is queried, a series of facts are added to the Proteomic Knowledge Base as soon as the new records are added to the PDB tables in the relational database. Either when inserting in the PDB page or the Bulk page.

The system will go through a sample of the input list and easily assign the right type to the whole list. After identifying the ids as PDB structure ids, it will verify for each ID if there is already any information for it in the Proteomic Knowledge Base database. If there isn't it will create a new fact a save in the Proteomic KB database and then proceed to appending it to a Prolog file. If there is, it will just save the facts already in the Proteomic KB to the Prolog file.

The Prolog file has the name `proteomic_kb_<yyyyMMddhhmmss>.pl` and is saved in `files/output` folder and the full flow of the Proteomics case study can be seen in Figure 5.1.

To comply with Prolog language specificities all text values were either converted to lowercase or wrapped in single quotes.

The facts in the Proteomic Knowledge Base have the following formats:

```
protein(structure_id, keywords, description, number_of_chains)

helix(structure_id, conf_id, helix_id, beg_res_name, beg_chain_id, beg_res_num,
      beg_ins_code, end_res_name, end_chain_id, end_res_num, end_ins_code, helix_class,
      details, length)

sheet(structure_id, sheet_id, sheet_type, number_strands, details, Beg_NDB_res_name,
      Beg_NDB_Chain_ID, Beg_NDB_res_num, Beg_NDB_ins_code, End_NDB_res_name,
      End_NDB_Chain_ID, End_NDB_res_num, End_NDB_ins_code, Sense,
      H_bond_cur_str_NDB_atom_id, H_bond_cur_str_NDB_res_name,
      H_bond_cur_str_NDB_Chain_ID, H_bond_cur_str_NDB_res_num,
      H_bond_cur_str_NDB_ins_code, H_bond_prev_str_NDB_atom_id,
      H_bond_prev_str_NDB_res_name, H_bond_prev_str_NDB_Chain_ID,
      H_bond_prev_str_NDB_res_num, H_bond_prev_str_NDB_ins_code)

strand(strand_id, structure_id, seq_align_beg, seq_align_beg_ins_code, seq_align_end,
      seq_align_end_ins_code, one_letter_sequence, chain_id)
```

And here, we present some extracts of the facts related to the structure with id '7a5p':

```
protein('7a5p', ['SPLICING'], 'Pre-mRNA-processing factor 17', 1)
protein('7a5p', ['SPLICING'], 'U2 small nuclear ribonucleoprotein B', 1)
protein('7a5p', ['SPLICING'], 'U5 small nuclear ribonucleoprotein 200 kDa helicase',
      1)

helix('7a5p', 'HELX_P1', 'AA1', 'UNK', '8', 5, '?', 'UNK', '8', 25, '?', 1, '?', 21)
helix('7a5p', 'HELX_P2', 'AA2', 'UNK', '8', 27, '?', 'UNK', '8', 33, '?', 1, '?', 7)
helix('7a5p', 'HELX_P3', 'AA3', 'UNK', '8', 43, '?', 'UNK', '8', 51, '?', 1, '?', 9)
```

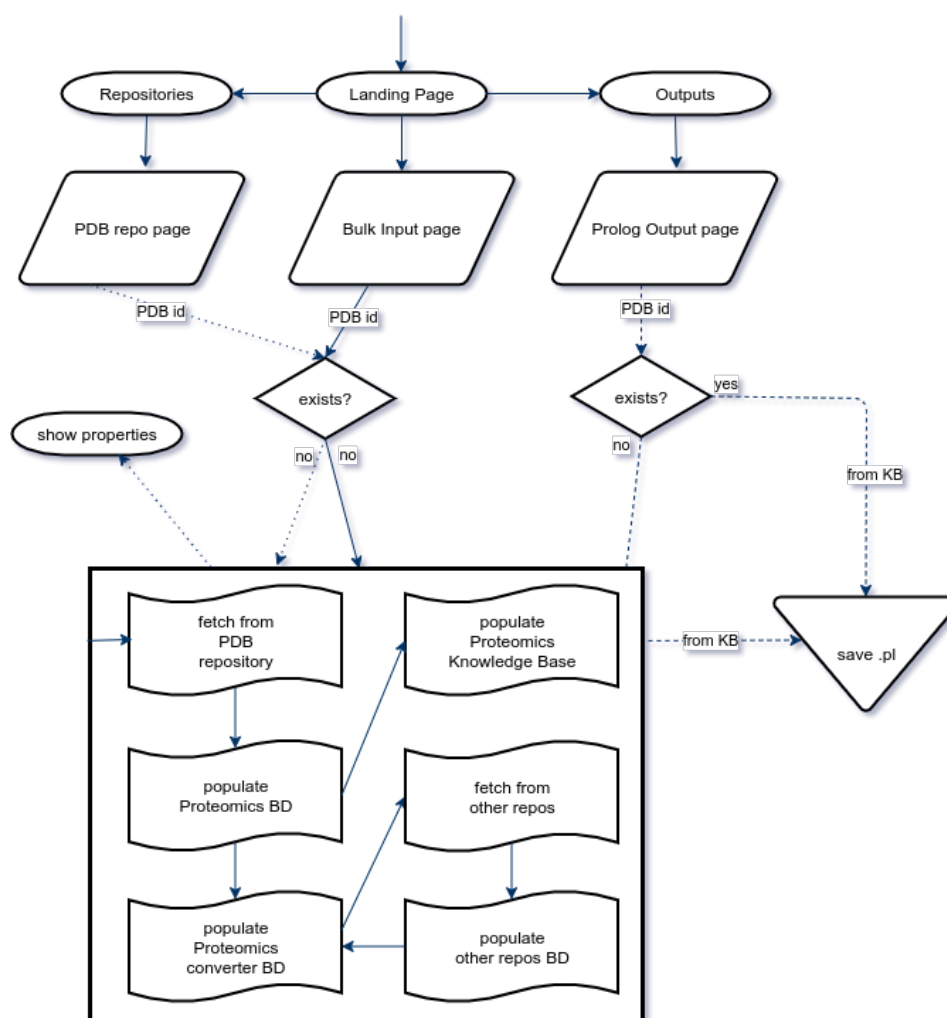


Figure 5.1: Proteomics case study pipeline

```

sheet('7a5p', 'AA1', '?', 4, '?', 'PHE', 'A', 1808, '?', 'PHE', 'A', 1810, '?', 'anti-
parallel', 'N', 'ASP', 'A', 1781, '?', 'O', 'PHE', 'A', 1808, '?')
sheet('7a5p', 'AA2', '?', 2, '?', 'ILE', 'A', 1777, '?', 'ASP', 'A', 1781, '?', '
parallel', 'O', 'ILE', 'A', 1862, '?', 'N', 'VAL', 'A', 1780, '?')
sheet('7a5p', 'AA3', '?', 8, '?', 'GLN', 'A', 1860, '?', 'VAL', 'A', 1863, '?', '
parallel', 'O', 'LYS', 'A', 1885, '?', 'N', 'ILE', 'A', 1861, '?')

```

```

...
strand('i', '7a5p', 1, '?', 504, '?', '
  mslicsisnevpehpcvpsvsnhvyerriekyiaengtdpinnqplse
  eqlidikvahpirkppsatsipalkalqdnewdavmlhsftlrqqqttrqelshalyqhaacrviarltkevtareal
  atlkpqaglivpqavpssqpsvvgagepmdlgeivgmtpeiiklqdkatvltterkkrqktvpeelvkpeelskyrqvash
  vglhsasipgilaldlcpstnkiltg\ngadknvvvfdksseqilatlkghtkkvtsvvhpsqdlvlfspdatiriwsvp
  nascvqvrahesavtglslhatgdynllssddqywafsdigtgrvltkvtddetsgcsltcaqfhpdglifgtgtdmsqiki
  wlkertnvanfpghsgpitsiaf\nsengyylataaddssvklwldrklknfktlqldnnfevkslifdqsgtylalggtdv
  qiyickqwteihftehsglttgnvafghhakfiastgmdrslkfysl', 'j')
...

```

5.2 Handling Chemistry

The testing dataset was obtained from two HIV-related study files with f(orward) and n(egative) examples. We have plucked the unique compound ids from these files and pasted them in the Bulk Inputs page, thus populating the Pubchem tables and the Chemistry conversion table. As this takes a while to process 39363 ids, we wait until the server informs us that the processing has finished.

We chose to first populate the Pubchem database and then request the PaDEL descriptors as this last phase takes too long.

Chemistry Knowledge Base

To populate the Chemistry Knowledge Base, we need to supply Pubchem compound ids in the Outputs/PaDEL page. We supplied the same ids already used to populate the Pubchem database.

In the backend, the details are obtained from the Pubchem tables and facts are generated and saved both to the knowledge base and to a Prolog file. If there are already facts for that Compound ID in the Chemistry KB, the system will just transfer them directly to a new Prolog file. Once again, all text values were either converted to lowercase or wrapped in single quotes.

The full flow of the Chemistry case study can be seen in Figure 5.2.

The Prolog file has the name `chemistry_kb_<??>_<yyyyMMddhhmmss>.pl` and is saved in `files/output` folder. The `<??>` can be either '2D', '3D' or 'fp', if the file refers to 2D, 3D descriptors or fingerprints, respectively. The smiles, atoms and bonds facts are saved to a file without the `<??>` part.

The facts in the Chemistry Knowledge Base have the following formats:

```
smiles(cid, smiles)

atom(cid, element, position)

bond(cid, element1, element2, bond_order)

descriptor(cid, descriptor, value)
and
descriptor(cid, fingerprint, binary_value)
```

Here are some examples of facts for fingerprints, 3D descriptors and 1D/2D descriptors for the molecule with Pubchem Compound ID 4280:

```
descriptor(4280, 'PubchemFP2', 0)
descriptor(4280, 'PubchemFP1', 1)
descriptor(4280, 'PubchemFP0', 1)

descriptor(4280, 'Ds', 1.45892185832188)
descriptor(4280, 'Ks', 0.6197710509438344)
descriptor(4280, 'Vs', 58.02765906734143)

descriptor(4280, 'Zagreb', 90.0)
descriptor(4280, 'XLogP', -2.0260000000000002)
descriptor(4280, 'WPOL', 29.0)
```

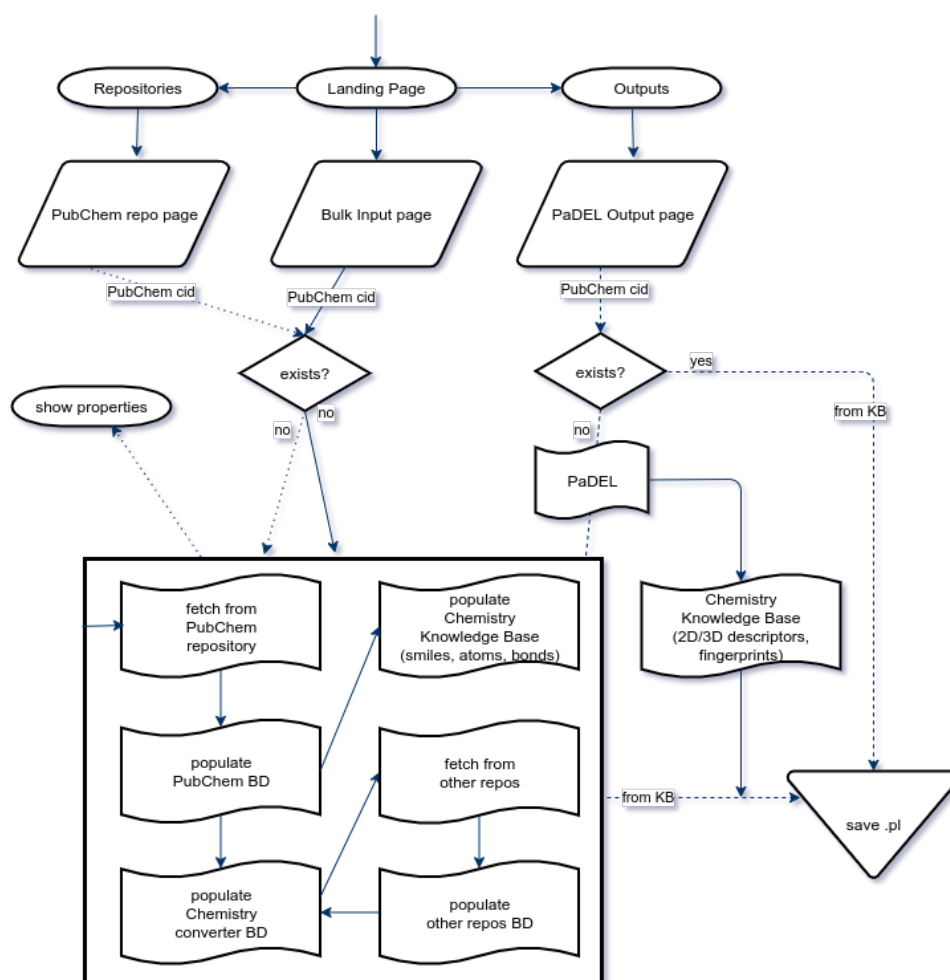


Figure 5.2: Chemistry case study pipeline

The following three sets of examples are of some of the fact for the atoms and bonds of the same molecule and, finally, the smiles fact.

```
atom(4280, 'h', 34)
atom(4280, 'n', 8)
atom(4280, 'o', 2)
```

```
bond(4280, 'o', 'h', 1)
bond(4280, 's', 'c', 1)
bond(4280, 's', 'n', 1)
bond(4280, 's', 'o', 2)
```

```
smiles(4280, 'c1=cc(=cc(=c1)s(=o)(=o)n(cco)cco)c(=o)o')
```

The PaDEL clauses take much time to fetch and save. We have tried different approaches trying to make the process more robust and speedy at the same time. We are talking of about 1444 descriptors and 881 fingerprints, which total 2325 lines per molecule.

At first, we thought we should ask the descriptors for all the molecules asked for at once. This method required just one API request and would save us time with inputs with dozens of thousands of compound ids. All ids at once could be impossible to fit in a single request, so the solution was to request the API for blocks with some molecules. This prompted another problem. When a subprocess call (to handle a particular molecule) at the PaDEL server timed out, how could we handle it appropriately if we did not know which molecule caused the error? We could subdivide further and try one-by-one in such faulty blocks. However, we would be wasting time. The better solution was to request descriptors and deal with each response one at a time.

Some larger molecules take an awful amount of time to get the descriptors, so we had to increase the request timeout to hundreds of seconds. We opt to signal the molecules that we are still not able to fetch the descriptors or fingerprints after that timeout.

We concluded that it would also be better to separate the Prolog output into three different files, one for fingerprints, another for 1D/2D descriptors, and the last for 3D descriptors.

There are two possible undesired outcomes in this procedure. One is that the molecule asked for still does not have a defined smiles notation registered. Without the smiles notation, we cannot obtain the descriptors from PaDEL. The other possibility is that the descriptors cannot be calculated on time, or there are still no fingerprints available for the compound.

The user is informed if any of the above occurs. The mail sent to the user informing that the process has finished, also details from which molecules it was not possible to get the information from.

5.3 Handling Genes

To populate the Genbank, we used a list with 19249 stomach cancer gene ids available in the format `name_id` separated by commas in a single line. We extracted the ids to a new file with just the ids, one per line. This new file was fed into the Bulk Page (preceded by a line with 'gen' to inform the backend which type of identifiers we were feeding it).

Besides the population of the Genbank database and the Gene table in particular (the other tables are also populated through their relationships), the calling to the conversion option also populates the Converter tables and the tables in the Kegg and Ensembl databases.

Genomic Knowledge Base

To populate the Genomic Knowledge Base, we need to supply Gene ids in the Outputs/Prolog page. We used the same list of ids used to populate the Genbank (and related) databases. In this case, we do not need to precede the ids with any indication of what they refer to. Since this endpoint only accepts PDB ids or Gene IDs, the parser at the backend will look through a small sample of given identifiers, and if it verifies that at least one of them corresponds to a PDB structure, it will treat all of them as such and as gene ids otherwise.

Naturally, if the identifier is already in the database, the system uses that record and only requests the repositories' API if otherwise.

In the backend, all gene-related tables are used to extract information to produce the inductive logic clause. From Gene, Protein and Ontology tables of the Genbank database, we collect information about the chromosome, the map_location, the start and end of the codification, the number of exons, the biotype, the category, the list of tissues and extra properties. We also get which go ids we have of each type. From the Gene, Protein and Transcript tables of the Ensembl database, we fetch the strand orientation, the begin, end and name of sequence region, the number of transcripts, the number of proteins, and the protein_accessions. From the Chemistry conversion table, PDB structures.

From the Kegg database, we gather data about amino-acid sequences from the Gene table, efficacies, related compounds, sequencing, diseases and classes from the Drug table, orthologies, more go ids, more related compounds and classes from Pathway, perturbants, metabolites, classes and types from Network, categories, pathogens, carcinogens and related genes from Disease.

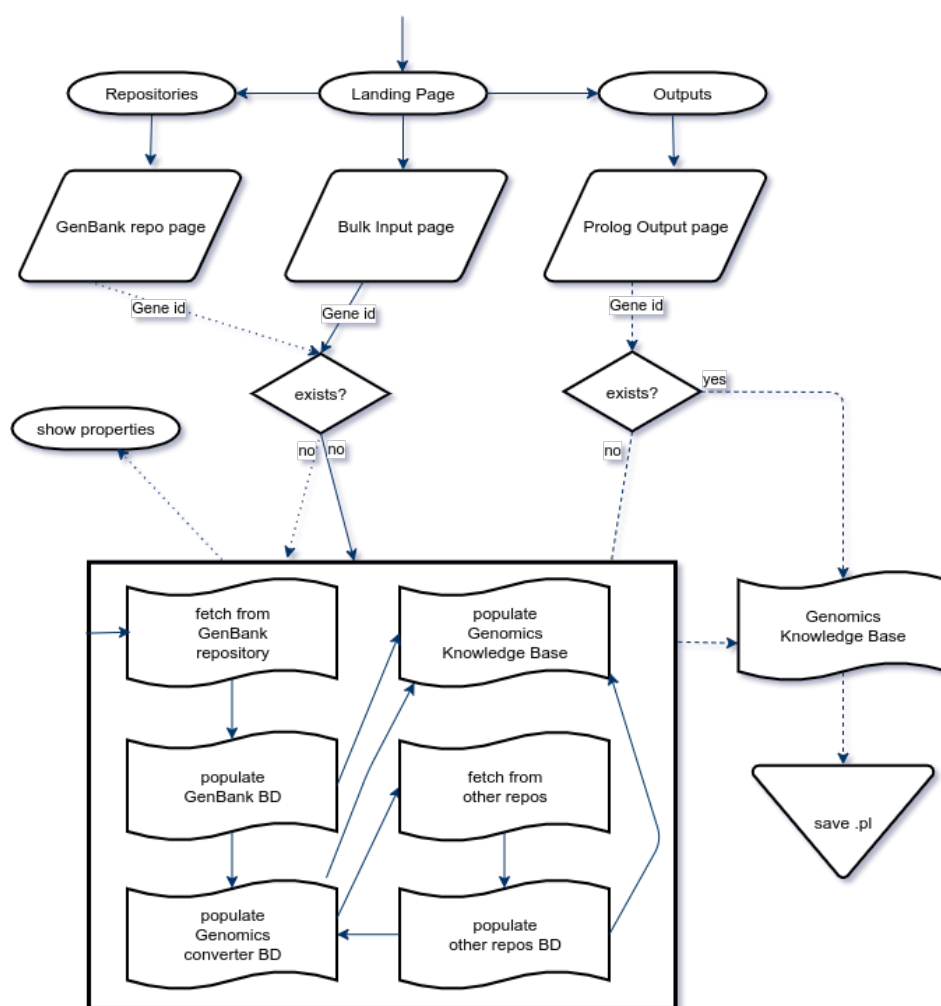


Figure 5.3: Genomics case study pipeline

One meaningful piece of information was still missing. It is important to know if a gene is located on the cell nucleus or in a mitochondrion as mitochondrial genes are very much linked to cancer because cancer demands large quantities of energy, and the energy is produced in the mitochondria. To that effect, we have added to the system a file with the list of mitochondrial genes that we have obtained from experts in the field. The list is checked for each gene the user asks to generate facts from, and a value added to the gene clause.

Once again, before creating the fact, the system verifies if it already has facts related to the provided Gene id in the Genomic KB database. If there isn't it will execute all the steps that lead to the creation of a new fact and saving it in the Genomic KB database and then proceed to appending it to a Prolog file. If there is, it will just save the facts already in the Genomic KB to the Prolog file.

The Prolog file has the name `genomic_kb_<yyyyMMddhhmmss>.pl` and is saved in `files/output` folder and the full flow of the Genomics case study can be seen in Figure 5.3.

Each Gene fact has the following format:

```
Gene(gene_id, mitochondrial, gene_chromosome, gene_map_location, start, stop,
gene_exon_count, gene_type, gene_category, gene_tissue_list, gene_prop,
seq_region_start, seq_region_end, seq_region_strand, seq_region_name,
gene_transcripts_count, gene_proteins_count, gene_proteins_accessions,
gene_gos_function, gene_gos_component, gene_gos_process, gene_pdb_ids,
gene_pathways, gene_diseases, gene_elements, gene_networks, gene_modules,
gene_keggorthologies, gene_drugs, gene_drugs_efficacies, gene_drugs_pubchems,
gene_drugs_diseases, gene_diseases_categories, gene_diseases_genes,
gene_diseases_orthologies, gene_disease_pathogens, gene_disease_carcinogens,
gene_drug_sequences, gene_drug_high_sequences, gene_gene_drug_low_sequences,
gene_drug_classes, gene_networks_type, gene_networks_pathways,
gene_networks_diseases, gene_networks_genes, gene_networks_perturbants,
gene_networks_metabolites, aaseq)
```

For the Amino-Acid sequence we use the one-letter encoding (in lowercase). To comply with Prolog requirements all text values were either converted to lowercase or wrapped in single quotes.

An example of a fact in the Genomic Knowledge Base (for gene with id 2):

```
gene(2, nuclear, 12, 12p13.31, 9067707, 9116228, 37, 'protein-coding', 'Broad
expression', [lung, urinary bladder, gall bladder, liver, fat, placenta,
endometrium, colon, kidney, prostate, spleen, heart, esophagus, adrenal, small
intestine, appendix, thyroid, brain, duodenum, stomach, testis, lymph node, ovary,
], '', 9067664, 9116229, -1, 12, 13, 5, ['NP_001334352', 'XP_006719119', '
NP_001334354', 'NP_000005', 'NP_001334353'], [48306, 19838, 19966, 19959, 43120],
[62023], [1869, 48863], ['2p9r', '4acq', '1bv8'], ['hsa04610'], [], [], [], [], ['
k03910'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [],
mgknkllhpslvllllvllptdasvsgkppqymvlvpsllhtettekgcvllsylvlnetvtsaslesvrgnrslftdleandv
lhcavafvpksssneevmltvqvkgptqefkkrvtvmvknedsllfvqtdksiykpgqtkfrvsvmdenfhplneliplvy
iqdpkgnriaqwsfqlegglkqfsfplssepfqgsykvvvqkksgrtehpftveefvlpkfevqvtvpkiitileemnvs
vcglytygkpvpghtvsvicrkysdasdchgedsqafcekfsgqlnshgcfyqqvktkvfqlkrkeyemklhteaqieegt
vcltrqasseitrtitklsfvkvdshfrqgipffgqvrldvgkvpipnkvivifirgneanyysnattdehglvqfsintnvm
gtsltvrnykdrspcygyqwvseeheeahhtaylvfspksfvhlepmshepcgthqtqvahyilnggtllglkklksfyyl
imakgdivrtgthglvkdqemkghfsisipvksdiapvarlliyavlpdgdvigdsakydvenclankvdlfspsqslpas
hahlrvtaappqsvcalravdqsvllmkpdaelsassvynllpekdlgtgfpplndqndedcinrhvnyingitytpvsstnek
dmysfledmglkaftnskirpkmpqlqqyemhgpeglrvgyesdvmggrharlvhvephtetvrkyfpetwiwldlvvn
```

```
sagvaevgtvpdtitewkagafclsedaglgisstaslralfqpfvltmpysvirgeaftlkatvlnylpkcirvsqlea  
spaflavpvekeqaphcicangrqtvswavtpkslgnvnftvsaealesqelcgtevpsvpehgrkdtvikpllvepegleke  
ttfnslpcsggevseelsklppnvveesarasvsvlgdilgsamqntqnllqmpygcgeqnmvlfapniyvldylnetqql  
tpeikskaigylnthyqqrqlykhydgsystfgerygrnqgntwltafvlktfaqarayifideahitqaliwlsqrqkngc  
frssgsllnnaikggvedevtlayitialeipltvthpvvrnalfclesawktaqegdhgshvytkallayafalagnqdk  
rkevlkslneeavkknsvhwerppkpkapvghfyepqapsaevemtsyvllyltaqpaptsedltsatnivkwitkqnaq  
ggfsstqdtvvalhalskygaatftrtgkaaqtisqsgtfsskfvdnnrlllqqvslpelpgeysmkvtgegcvtlqtsl  
kynilpekeefpfaigvqtlpqtcdpkahstsfqislsvsytgsrsasnaividvkmvsgfiplkptvkmmlersnhvsrtevs  
snhvliylkvsnqtlsffvtvlqdvprdlkpaivkvydyetdefaiaeynapcskdlgna)
```

5.4 Summary

In this chapter we have explained how we have used the system to handle problems in three different fields: Chemistry, Genomics and Proteomics. We described which inputs we used, how we fed them to the system and the output the system returned.

Chapter 6

Conclusions and Future Work

In this concluding chapter, we present the results of this project, draw conclusions and point to directions for future work.

6.1 Conclusions

The main goal of this dissertation was to produce a system to access bioinformatics and cheminformatics repositories, enrich the data by generating pieces of information aggregating details from different sources and finally creating files with this enriched information to feed to ILP systems.

This was accomplished. With this solution one can get enriched genomic, proteomic and chemistry information simply by introducing ids in the Web Application interface and use the outputs in ILP systems to obtain data mining classifications. We have already produced three datasets: One with around twenty thousand gene clauses from as many cancer gene ids, another with almost one and a half million proteomic clauses from 45 thousand HIV-related Pdb structure ids and a third with almost three and a half million chemistry clauses from almost 40 thousand compound ids.

The clean looking and appealing interface is easy to use and meets all the needs in terms of inputs, outputs and functionalities. It is no longer necessary to go into different repositories nor to do conversions. All is available in the same portal making all tasks faster and less laborious.

Regarding the four FEUP projects mentioned in state of the art, our application works with more repositories, five in all, allowing a broader scope, something requested in those theses. It is not limited to genomics, allowing outputs for chemistry and proteomics studies since it is also the first of these projects to deal with chemistry. Also requested in those theses was the possibility of exporting to more file types, which we also achieved. Those projects are more about data integration, while our project is mainly focused on data enrichment and Prolog fact production. It does not have integrated Machine Learning / Data Mining algorithm processing, but it is only necessary to provide the outputs to systems like Weka or Aleph.

The other projects mentioned in state of the art do not enrich either. They are more of analysis, classification and clustering, which can be done from the outputs generated we generate.

It was a good decision to choose Python/Django as many libraries are easily available, the development is easy, and new blocks are easily isolated tested. Furthermore, as some of the tables hold several million records, using PostgreSQL was also a great decision.

One of the biggest challenges was to find a suitable common property that could be used as a primary key for the Proteomic conversion table. Another problem lay with the reply formats from specific API. Some repository replies, namely from Kegg, may format blocks of information differently, causing problems when we get unexpected behaviours that need to be addressed even after reading the documentation and successfully obtaining testing samples. Other times, the REST API started replying differently and what was working good needs to be adapted.

Still, in the repository access domain, we needed to add robustness to the calls to handle downtimes or processing timeouts.

6.2 Future Work

If there was no time limit, we could gather data from at least one more Cheminformatics repository (ChEmbl) and another Proteomics (Uniprot).

Populating the facts with PaDEL fingerprints and descriptors takes a long time, and we would need more time to get a more representative dataset.

An improvement to be added in the future is a more elaborate user management scheme, separating data from different users. Each researcher that accesses the system will have his own data.

References

- [1] P. Basak. *A GUI For Defining Inductive Logic Programming Tasks For Novice Users*. PhD thesis, University of Minnesota Twin Cities, 2017.
- [2] D. A. Benson et al. GenBank. *Nucleic Acids Research*, 36(suppl_1):D25–D30, 12 2007.
- [3] D. A. Benson et al. Nucleic acids res. 2013 jan;41(database issue):d36-42. *Epub-Nov*, 27, 2012.
- [4] H. M. Berman et al. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 01 2000.
- [5] D. Bora and O. P. Gupta. A comparative study between fuzzy clustering algorithm and hard clustering algorithm. *International Journal of Computer Trends and Technology*, 10:108–113, 04 2014.
- [6] Q. Boucher et al. *Engineering Configuration Graphical User Interfaces from Variability Models*, pages 1–46. Springer, 10 2017.
- [7] F. K. Brown. Chemoinformatics: What is it and how does it impact drug discovery. In J. A. Bristol, editor, *Annual Reports in Medicinal Chemistry*, volume 33, pages 375 – 384. Academic Press, 1998.
- [8] M. S. Brown. What IT Needs To Know About The Data Mining Process, 2015.
- [9] T. Clark and M. Hicks. Models of necessity. *Beilstein Journal of Organic Chemistry*, 06 2020.
- [10] L. De Raedt. *Inductive Logic Programming*, pages 529–537. Springer US, Boston, MA, 2010.
- [11] W. J. Diniz and F. Canduri. Bioinformatics: an overview and its applications. *Genetics and molecular research : GMR*, 16:1, 2017.
- [12] L. Ertöz et al. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data*, 05 2003.
- [13] E. Frank et al. The WEKA Workbench. Online Appendix. In *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2016.
- [14] N. Friedman et al. Bayesian network classifiers. *Mach. Learn.*, 29(2–3):131–163, November 1997.

- [15] M. Gaurav. Docker Guide - Build a fully production ready machine learning app with React, Django, and PostgreSQL on Docker, 2020.
- [16] E. Giacomidis et al. Blind nonlinearity equalization by machine learning based clustering for single- and multi-channel coherent optical ofdm. *Journal of Light-wave Technology*, PP:1–1, 11 2017.
- [17] G. Gini. How far chemistry and toxicology are computational sciences? *Springer-Briefs in Applied Sciences and Technology*, 7:15–33, 06 2014.
- [18] O. P. Gupta and S. Rani. Bioinformatics Applications and Tools: An Overview. *Biometrics and Bioinformatics*, 3:107–110, jan 2011.
- [19] J. Han et al. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3rd edition, 2011.
- [20] D. M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. PMID: 14741005.
- [21] S. Heller et al. The iupac international chemical identifier. *Journal of Cheminformatics*, 7, 12 2015.
- [22] A. Holovaty and J. Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right, Second Edition*. Apress, USA, 2nd edition, 2009.
- [23] S. Jónsdóttir et al. Prediction methods and databases within chemoinfromatics: emphasis on drugs and drug candidates. *Bioinformatics (Oxford, England)*, 21:2145–2160, 2005.
- [24] M. Kanehisa. Toward understanding the origin and evolution of cellular organisms. *Protein Science*, 28(11):1947–1951, 2019.
- [25] M. Kanehisa et al. KEGG: integrating viruses and cellular organisms. *Nucleic Acids Research*, oct 2020.
- [26] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28(1):27–30, jan 2000.
- [27] A. Kassambara. Cluster Validation Statistics: Must Know Methods, 2020.
- [28] S. Kim et al. Pubchem in 2021: new data content and improved web interfaces. *Nucleic acids research*, 49((D1)):D1388–D1395, 2019.
- [29] R. D. King. Inductive logic programming: techniques and applications by nada lavrac and saso dzeroski, ellis horwood, uk, 1993, pp 293, £39.95, isbn 0-13-457870-8. *The Knowledge Engineering Review*, 9(3):311–312, 1994.
- [30] P. Larrañaga et al. Machine learning in bioinformatics. *Briefings in Bioinformatics*, 7(1):86–112, mar 2006.
- [31] N. M. Luscombe et al. What is bioinformatics? a proposed definition and overview of the field. *Methods Inf Med*, 40(4):346–58, 2001.
- [32] T. S. Madhulatha. An overview on clustering methods, 2012.

- [33] K. Martínez et al. The impact of chemoinformatics on drug discovery in the pharmaceutical industry. *Expert Opinion on Drug Discovery*, 15:1–14, 2020.
- [34] P. Martins et al. *Gene prediction using Deep Learning*. PhD thesis, Universidade do Porto, 2018.
- [35] MDN Contributors. Django introduction, 2021.
- [36] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.
- [37] S. Min et al. Deep learning in bioinformatics. *Briefings in bioinformatics*, 18(5):851–869, sep 2017.
- [38] J. B. O. Mitchell. Machine learning methods in chemoinformatics. *Wiley Interdisciplinary Reviews. Computational Molecular Science*, 4(4):468–481, 2014.
- [39] U. Mudunuri et al. bioDBnet: the biological database network. *Bioinformatics*, 25(4):555–556, 01 2009.
- [40] S. Narkhede. Understanding AUC - ROC Curve, 2018.
- [41] L. Natividade et al. *Data Mining para análise dos resultados de Gene Expression*. PhD thesis, Universidade do Porto, 2017.
- [42] A. Nightingale et al. The Proteins API: accessing key integrated protein and genome information. *Nucleic Acids Research*, 45(W1):W539–W544, jul 2017.
- [43] G. Paynter. Attribute-Relation File Format (ARFF), 2008.
- [44] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [45] C. Pereira et al. *A Toolbox for Genomic Studies*. PhD thesis, Universidade do Porto, 2019.
- [46] U. Raudvere et al. g:Profiler: a web server for functional enrichment analysis and conversions of gene lists (2019 update). *Nucleic Acids Research*, 47(W1):W191–W198, 05 2019.
- [47] W. Reese. Nginx: The high-performance web server and reverse proxy. *Linux J.*, 2008(173), September 2008.
- [48] M. Ritchie et al. Methods of integrating data to uncover genotype-phenotype interactions. *Nat Rev Genet*, 16:85–97, February 2015.
- [49] U. Roessner and J. Bowne. What is metabolomics all about? *BioTechniques*, 46(5):363–365, 2009. PMID: 19480633.
- [50] E. Sayers. A general introduction to the e-utilities. Entrez Programming Utilities Help [Internet]. *National Center for Biotechnology Information, Bethesda*, 2010.
- [51] C. Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 2000.

- [52] Shmula. Precision, Accuracy, Measurement System, 2010.
- [53] A. Srinivasan. The Aleph Manual, 2007.
- [54] D. Vanhuyse and R. Camacho. *Portal Web para enriquecimento de informação Genómica e Proteómica*. PhD thesis, Universidade do Porto, 2017.
- [55] H. Verli. *Bioinformática: da biologia à flexibilidade molecular*. Sociedade Brasileira de Bioquímica e Biologia Molecular, 2014.
- [56] W. Vorhies. CRISP-DM – a Standard Methodology to Ensure a Good Outcome, 2016.
- [57] G. Wang and R. Dunbrack. Pisces: a protein sequence culling server. *Bioinformatics (Oxford, England)*, 19:1589–91, 09 2003.
- [58] W. Warr. EXTRACT FROM 218TH ACS NATIONAL MEETING AND EXPOSITION NEW ORLEANS, LOUISIANA, AUGUST 22-26, 1999. Technical report, Wendy Warr & Associates, 2000.
- [59] S. P. Wathen. Introduction to cheminformatics for green chemistry education. *Physical Sciences Reviews*, 4(2):20180078, 2019.
- [60] D. Weichselbaum et al. Fuento: functional enrichment for bioinformatics. *Bioinformatics (Oxford, England)*, 33, 03 2017.
- [61] D. Weininger et al. Algorithm for generation of unique smiles notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101, 1989.
- [62] Wikipedia contributors. Kmeans-gaussian-data — Wikipedia, the free encyclopedia, 2011. [Online; accessed 25-January-2021].
- [63] I. H. Witten et al. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Amsterdam, 3 edition, 2011.
- [64] C. W. Yap. Padel-descriptor: An open source software to calculate molecular descriptors and fingerprints. *Journal of Computational Chemistry*, 32(7):1466–1474, 2011.
- [65] A. D. Yates et al. Ensembl 2020. *Nucleic Acids Research*, 48(D1):D682–D688, jan 2020.

Appendix A

Databases

PDB

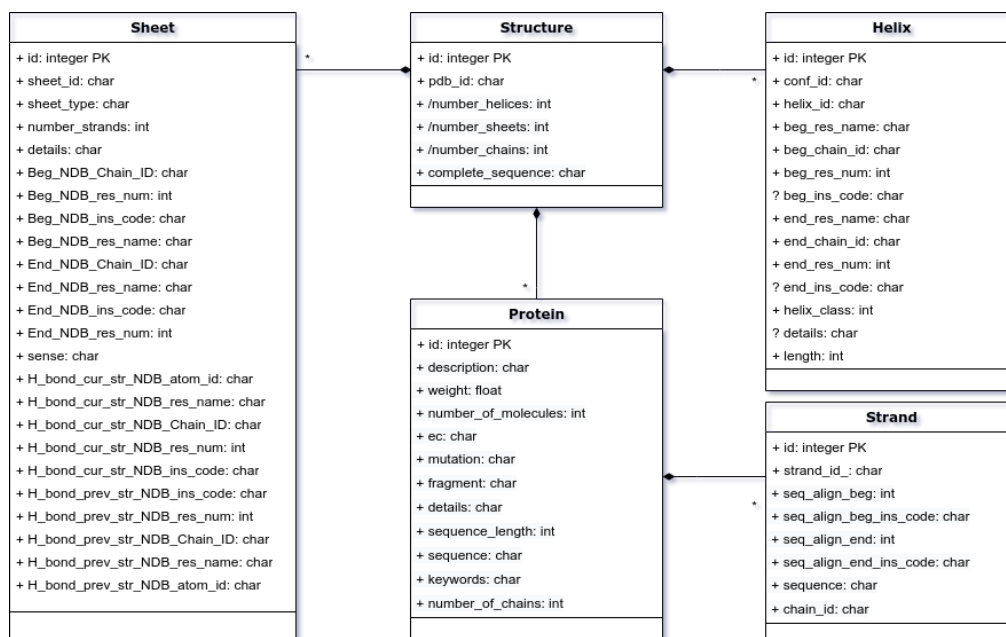


Figure A.1: UML of the conceptual data model of the PDB database

Pubchem

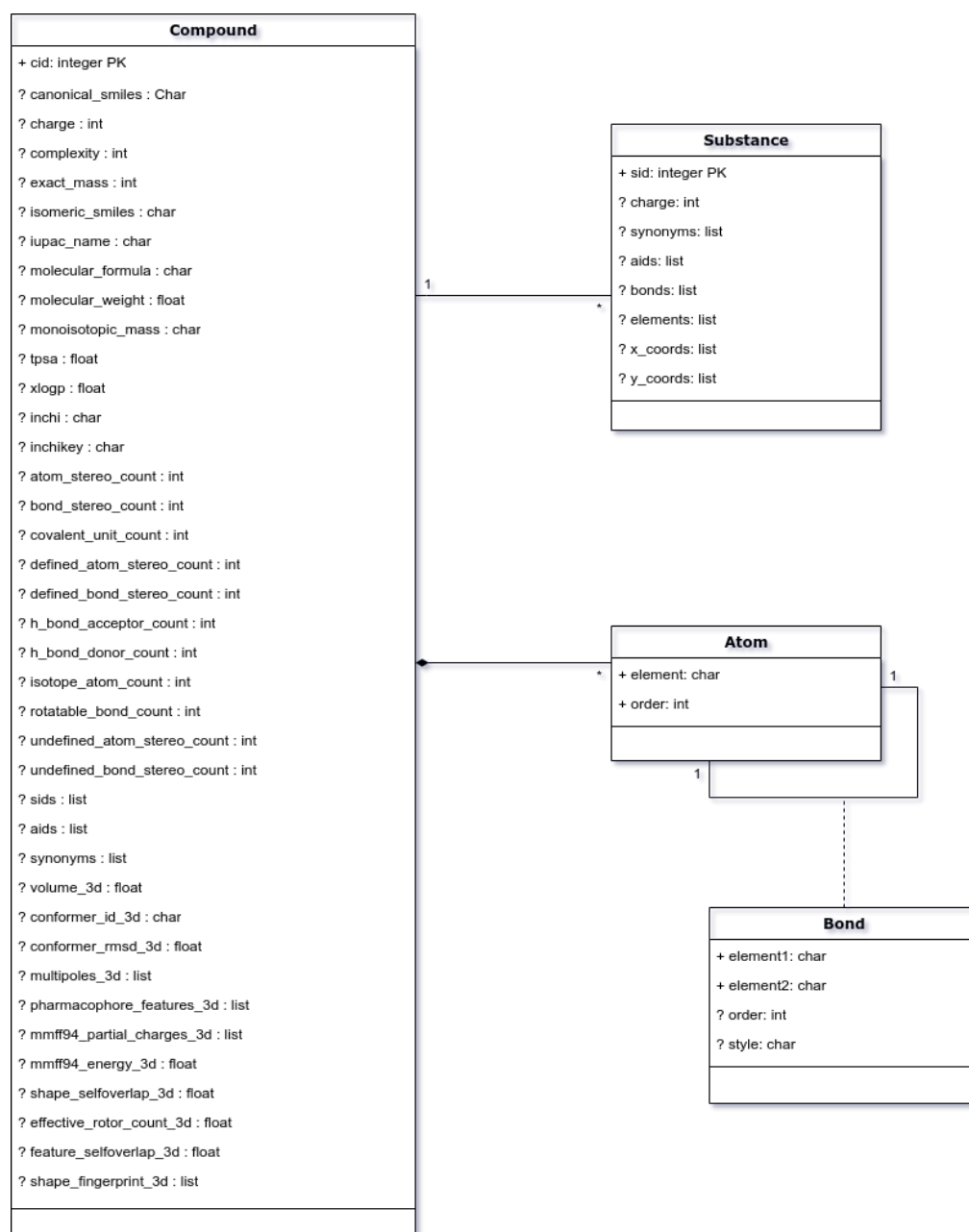


Figure A.2: UML of the conceptual data model of the Pubchem database

Kegg

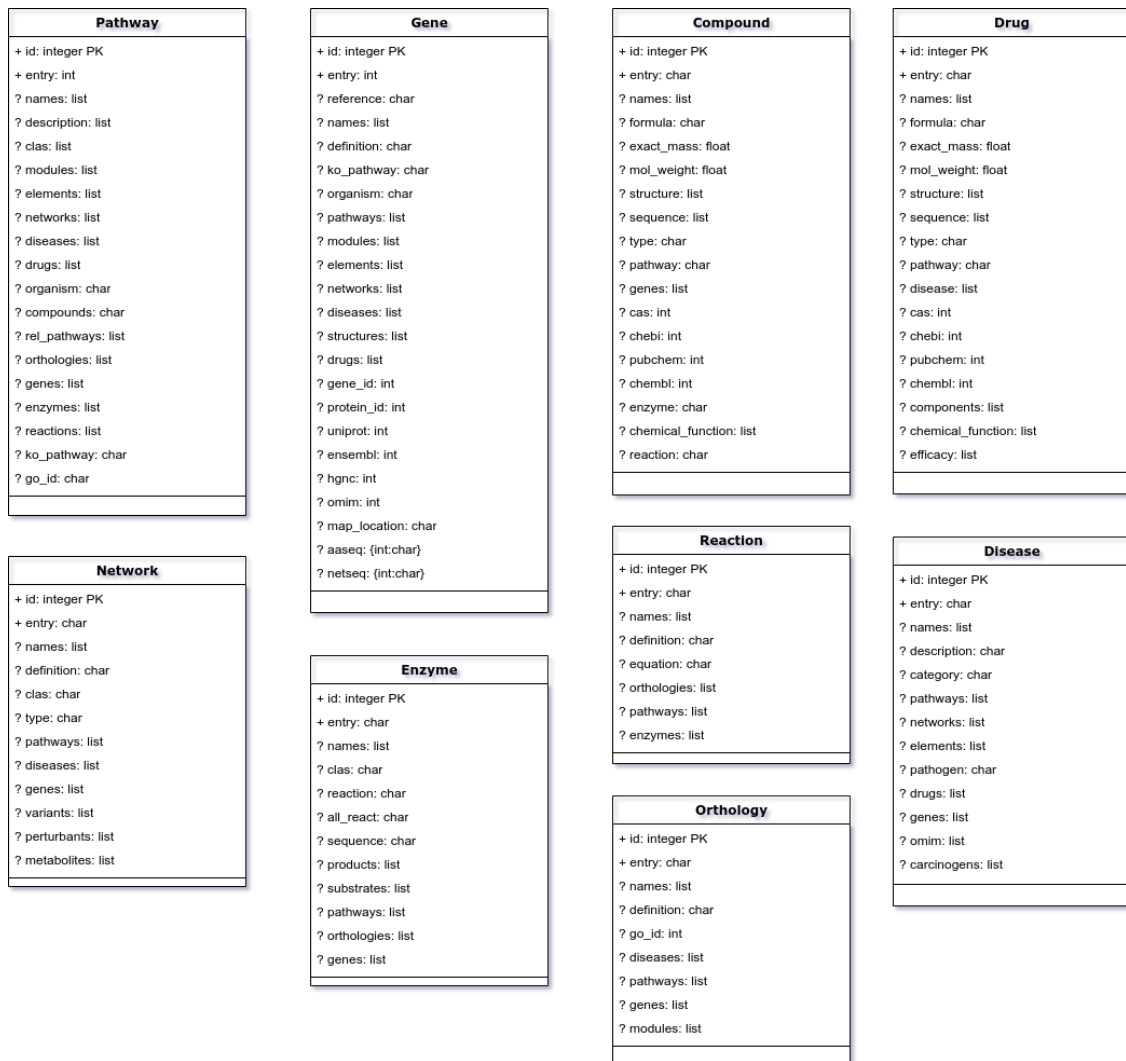


Figure A.3: UML of the conceptual data model of the Kegg database

GenBank

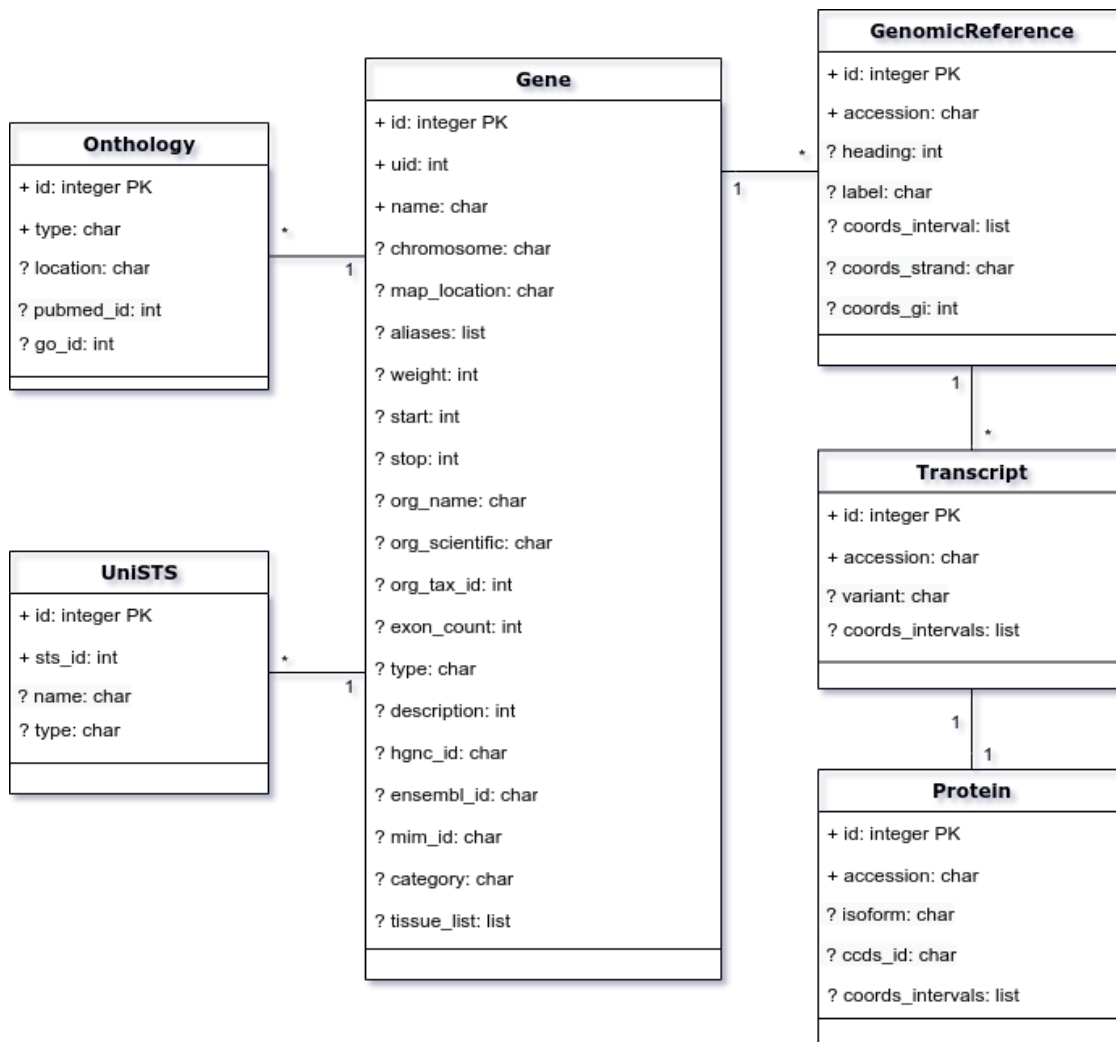


Figure A.4: UML of the conceptual data model of the Genbank database

Ensembl

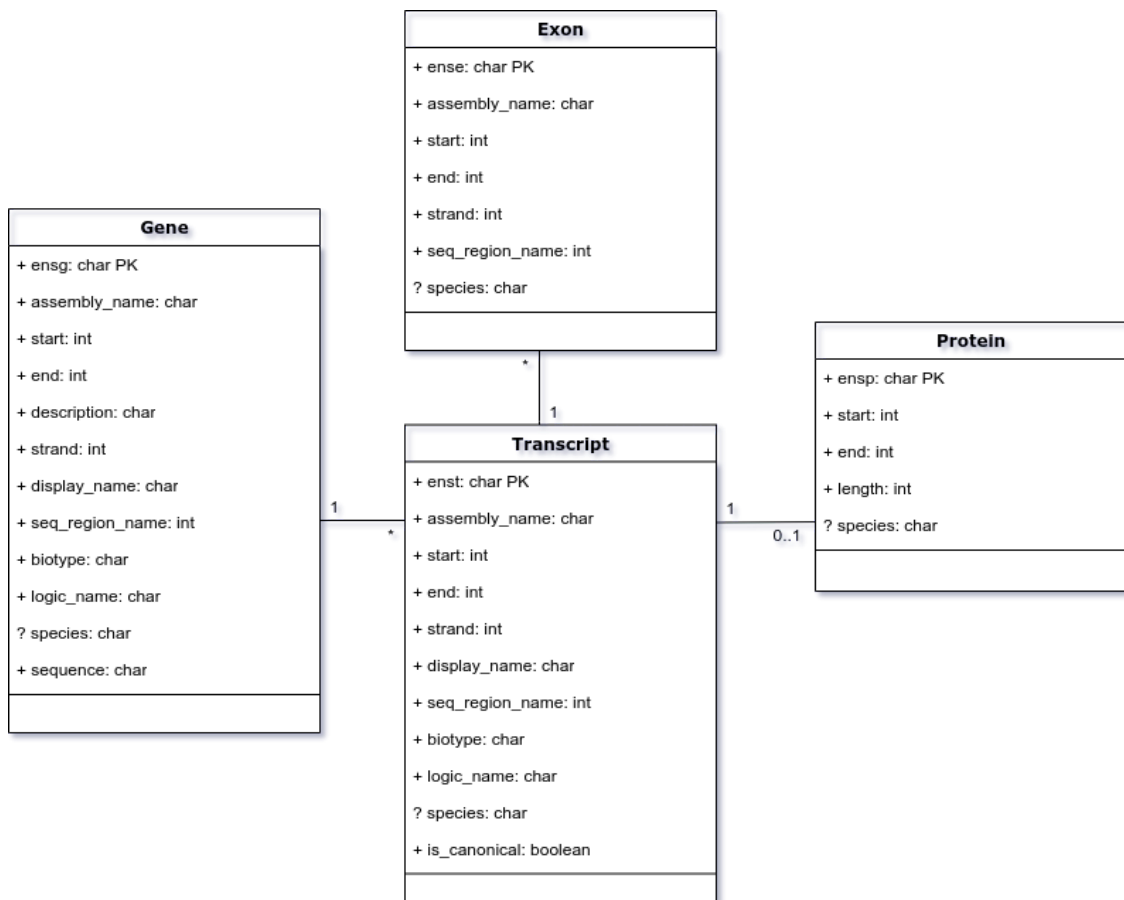


Figure A.5: UML of the conceptual data model of the Ensembl database

Go

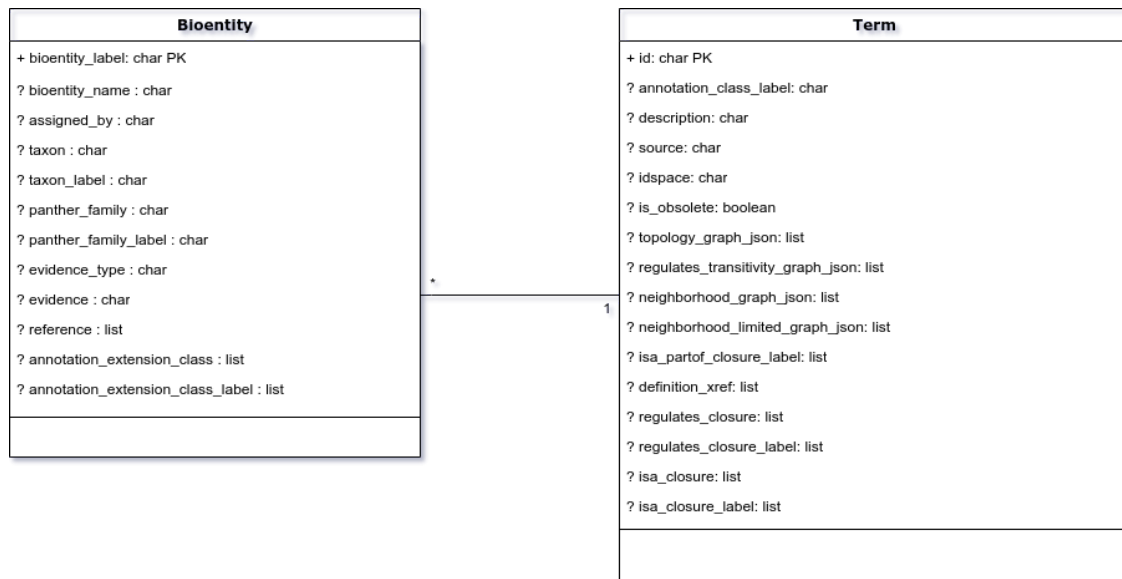


Figure A.6: UML of the conceptual data model of the Go database

Convert

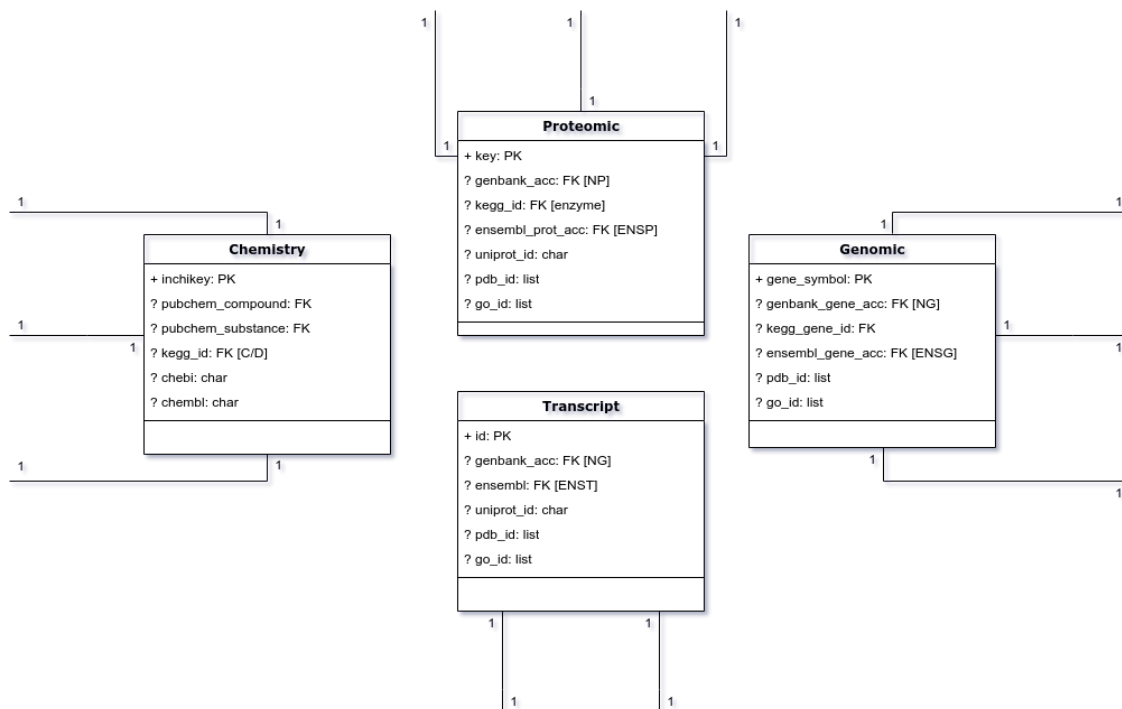


Figure A.7: UML of the conceptual data model of the Converter database

Knowledge Base

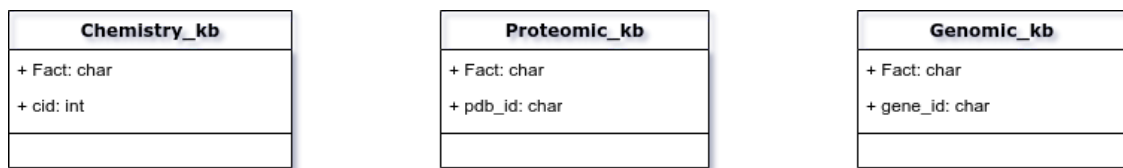


Figure A.8: UML of the conceptual data model of the Knowledge Base database
[each fact format is described in the Handling sections]

Appendix B

More interface pages examples

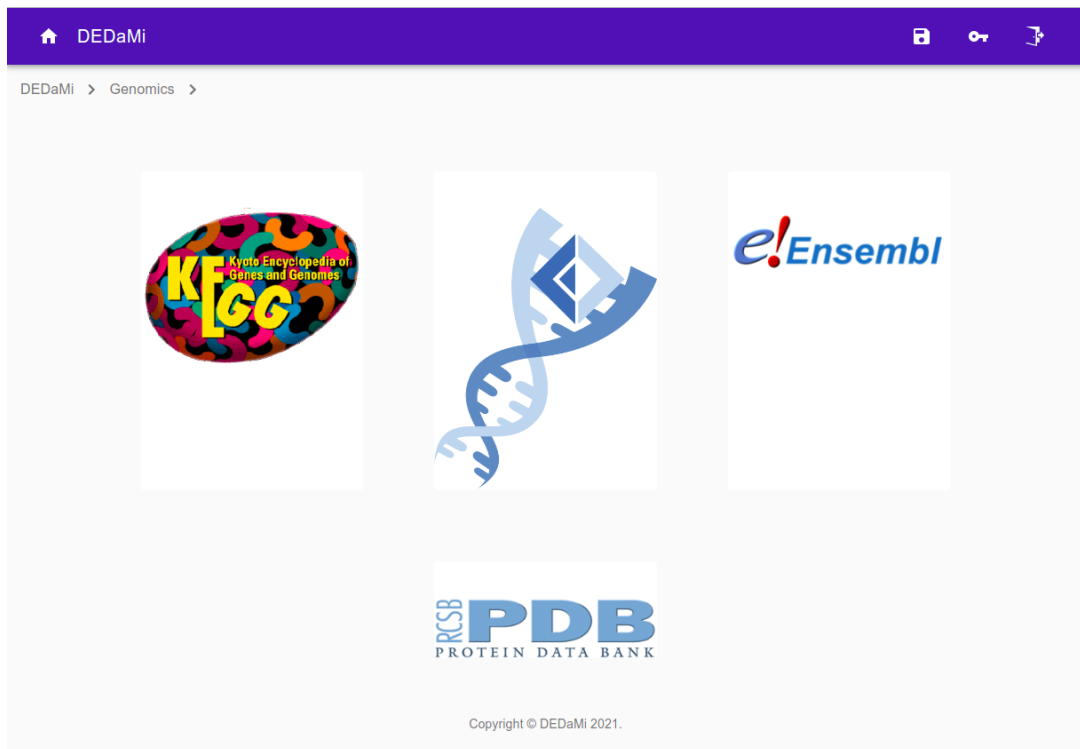


Figure B.1: Genomic field page

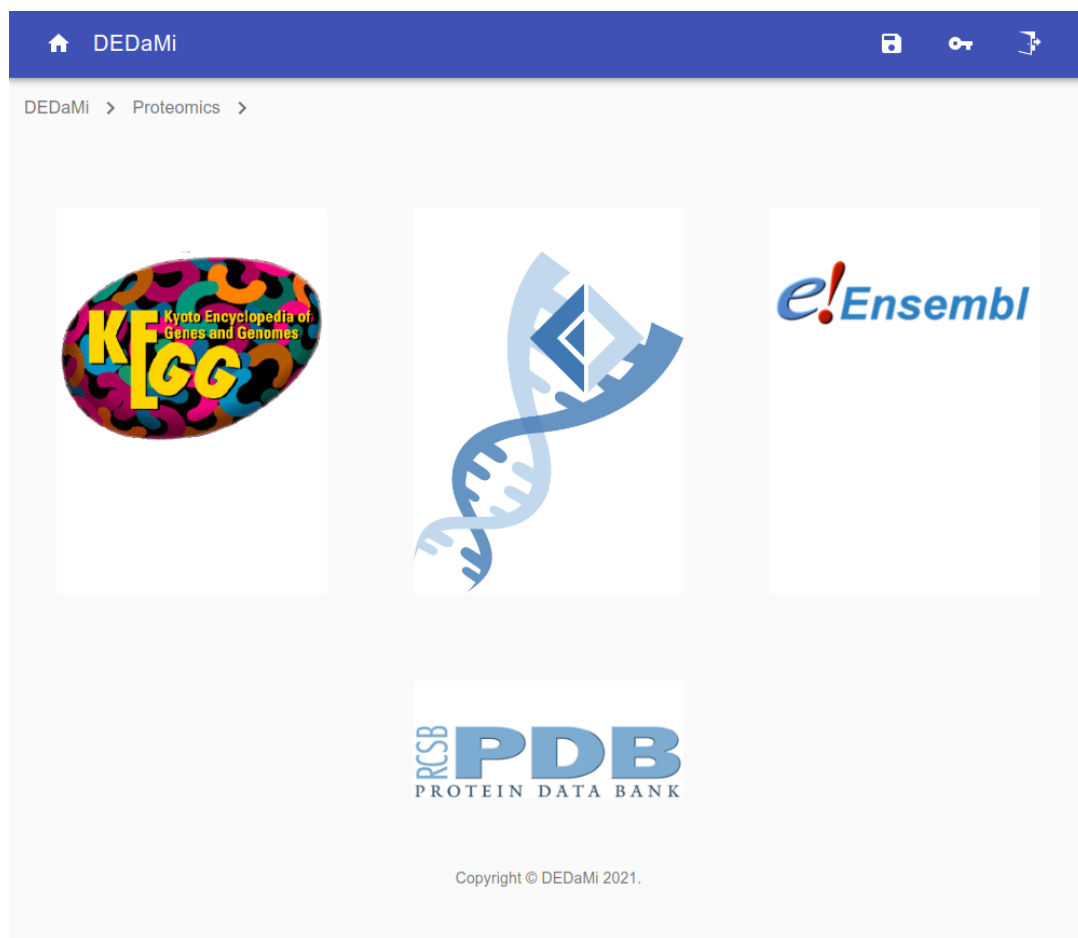


Figure B.2: Proteomic field page

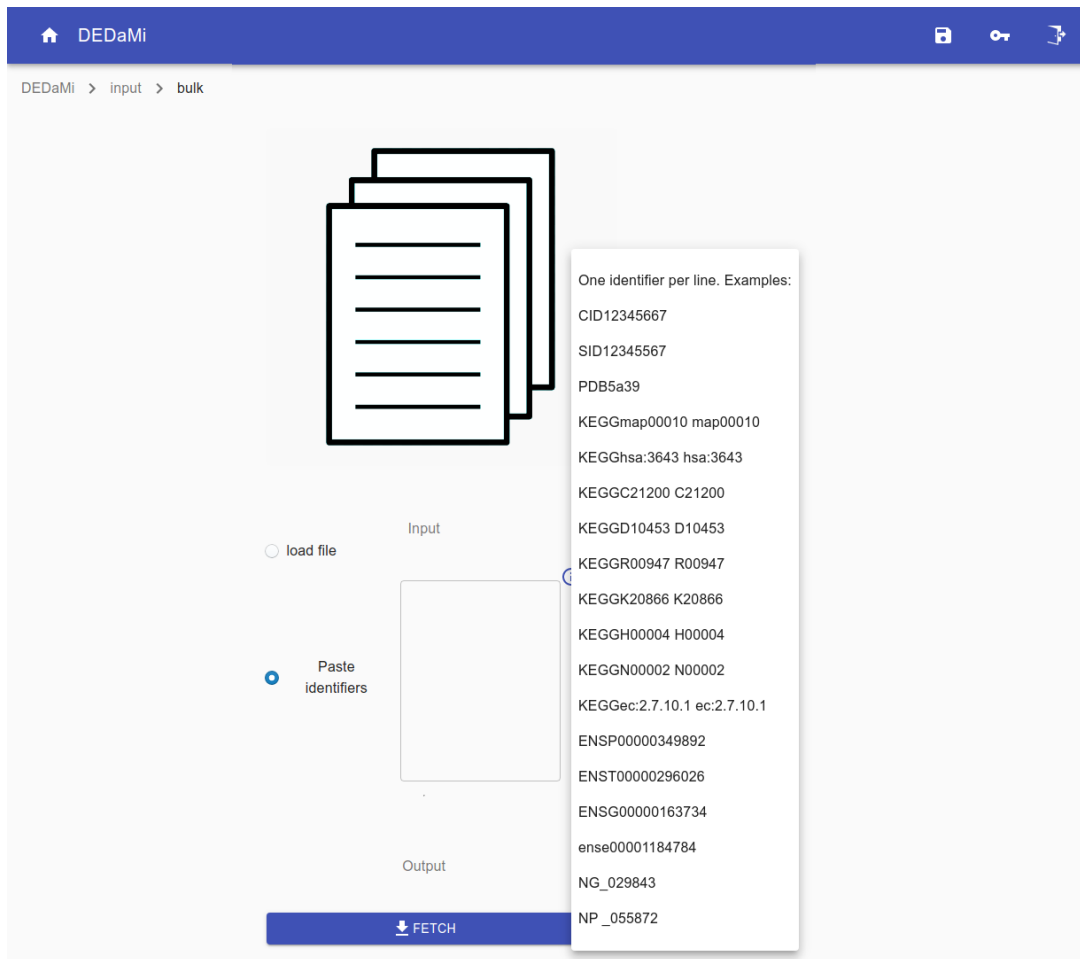


Figure B.3: Bulk Input page with help tooltip

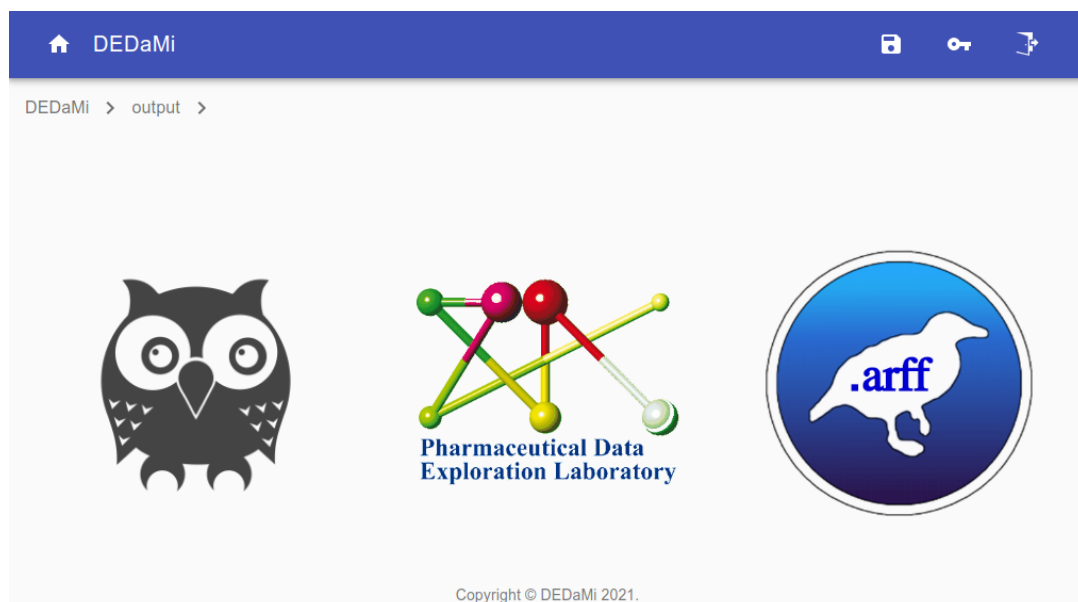


Figure B.4: Outputs page

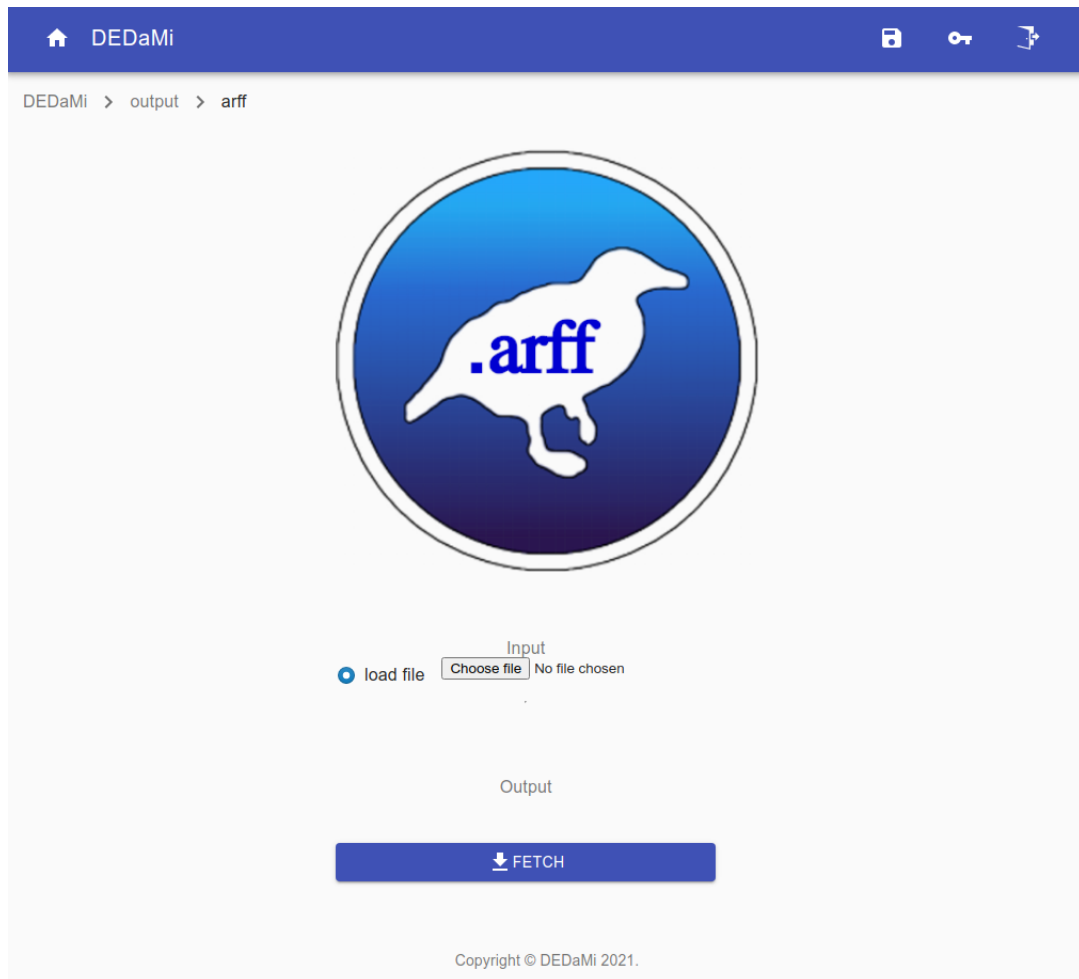


Figure B.5: CSV to ARFF output page

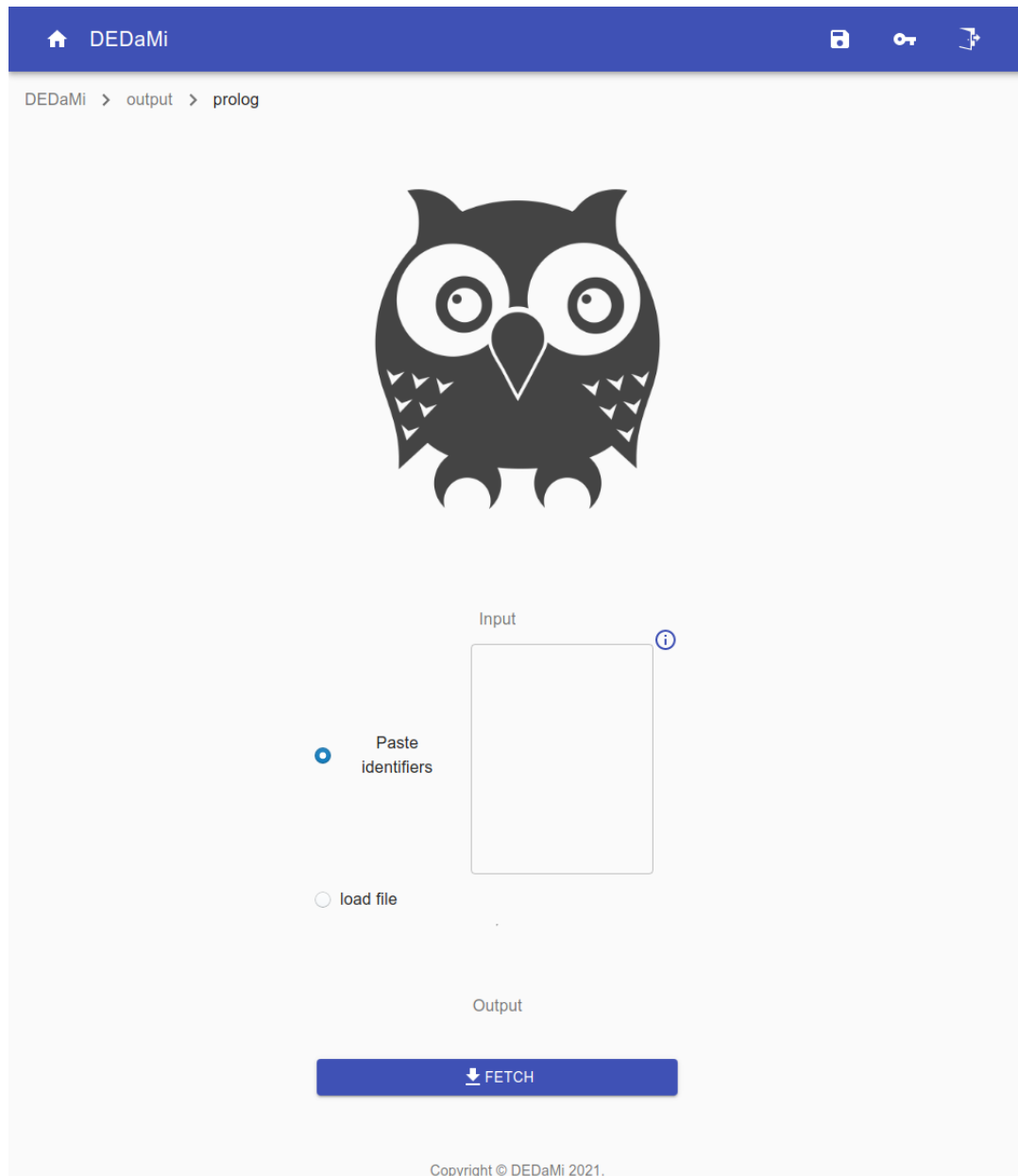


Figure B.6: Output Prolog page

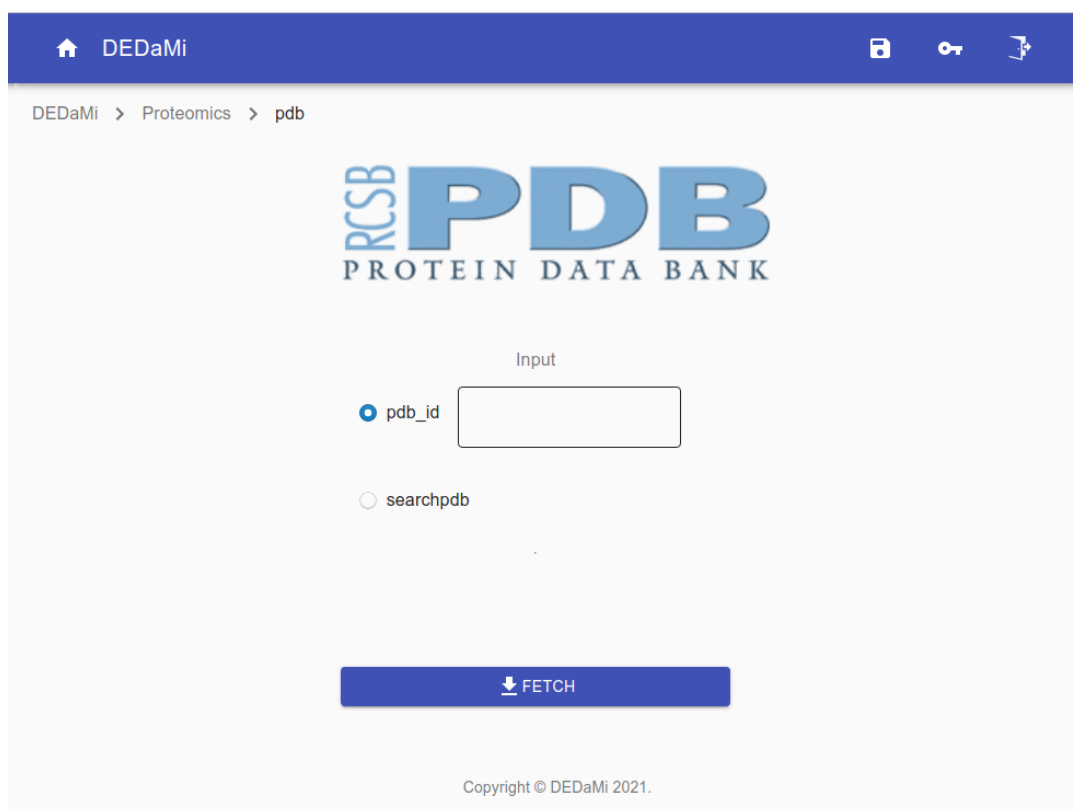


Figure B.7: PDB input page

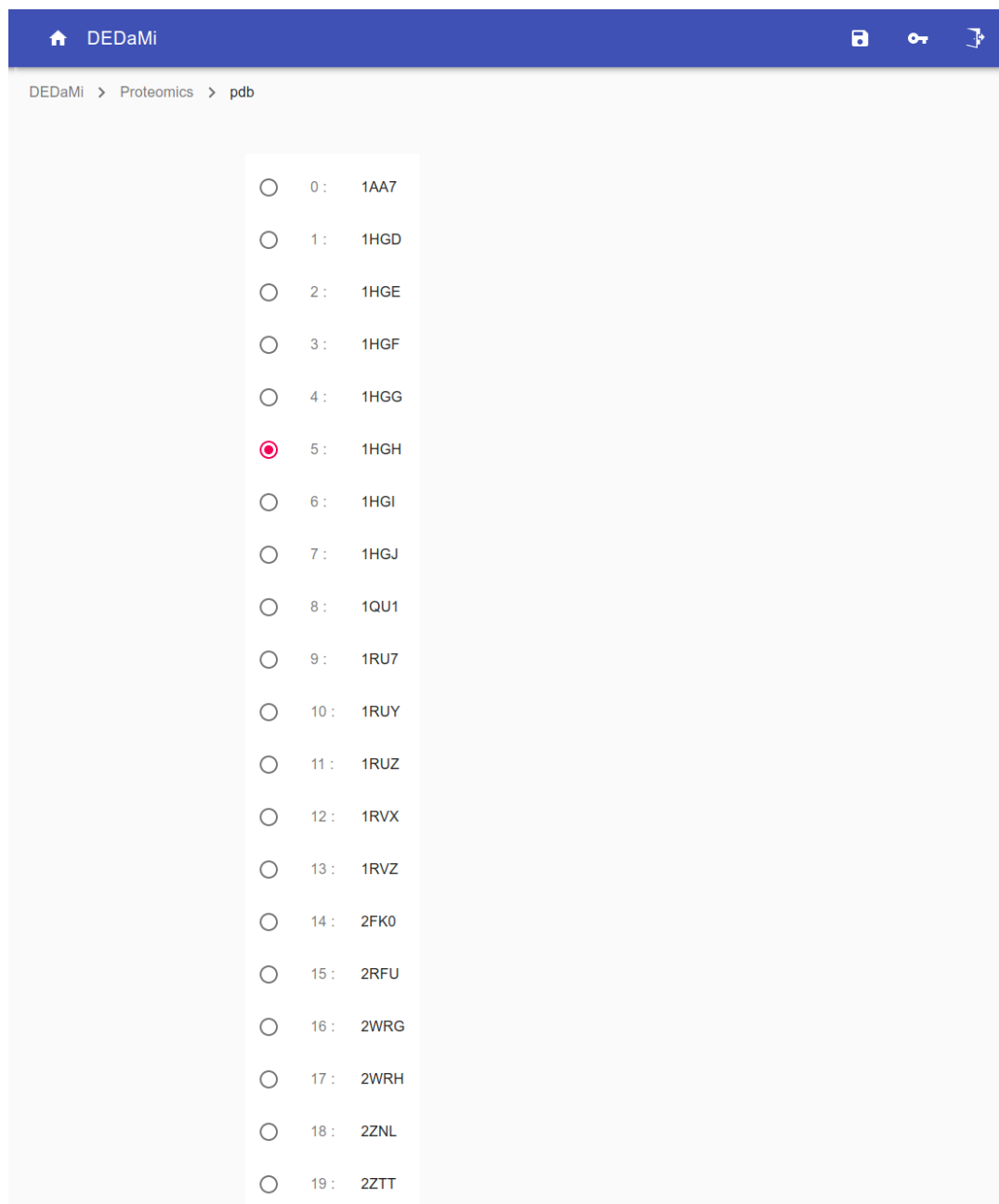


Figure B.8: PDB search results

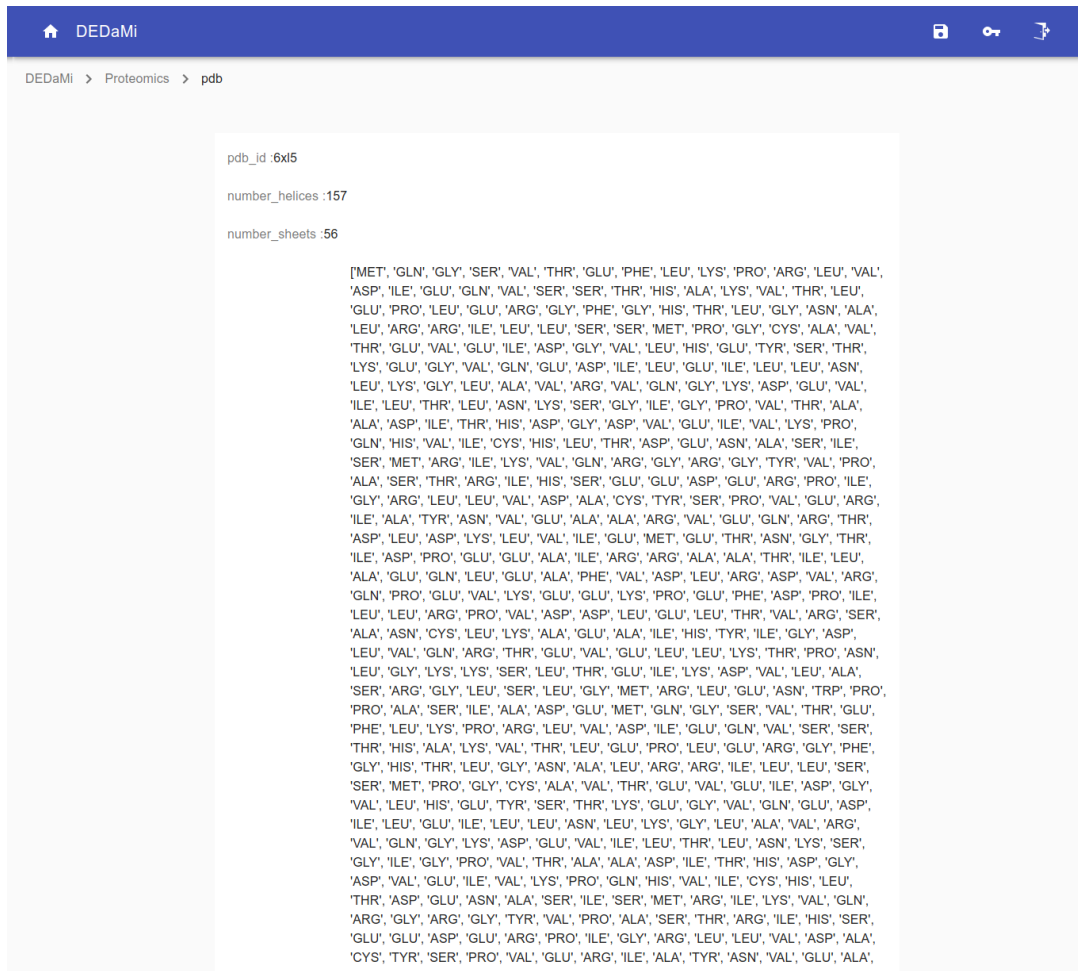


Figure B.9: PDB output page

DEDaMI

DEDaMI > Cheminformatics > pubchem

PubChem

2D 3D
Substance Compound

Input

cid

Name

Smiles

InChI

InChIKey

Formula

Output

All

Molecular Formula

Molecular Weight

Canonical SMILES

InChI

InChIKey

Isomeric SMILES

IUPAC Name

Synonyms

XLogP

ExactMass

MonoisotopicMass

TPSA

Complexity

Charge

HBondDonorCount

RotatableBondCount

HeavyAtomCount

IsotopeAtomCount

AtomStereoCount

DefinedAtomStereoCount

UndefinedAtomStereoCount

BondStereoCount

DefinedBondStereoCount

UndefinedBondStereoCount

CovalentUnitCount

Sids

Aids

Copyright © DEDaMI 2021.

Figure B.10: Pubchem repo page for compounds (2D output)

The screenshot shows a web interface for PubChem. At the top, there is a navigation bar with 'DEDaMI' and a breadcrumb trail 'DEDaMI > Cheminformatics > pubchem'. The main heading is 'PubChem' with a logo. Below the heading, there are two rows of radio buttons: '2D' (selected) and '3D', and 'Substance' and 'Compound' (selected). An 'Input' field is present with a dropdown menu set to 'cid'. Below the input field, there are radio buttons for 'Name', 'Smiles', 'InChI', 'InChIKey', and 'Formula'. Under the 'Output' section, there is a checked 'All' option and a long list of checkboxes for various molecular and physical properties, including Molecular Formula, Molecular Weight, Canonical SMILES, InChI, InChIKey, Isomeric SMILES, IUPAC Name, Synonyms, XLogP, ExactMass, MonoisotopicMass, TPSA, Complexity, Charge, HBondDonorCount, RotatableBondCount, HeavyAtomCount, IsotopeAtomCount, AtomStereoCount, DefinedAtomStereoCount, UndefinedAtomStereoCount, BondStereoCount, DefinedBondStereoCount, UndefinedBondStereoCount, CovalentUnitCount, Sids, Aids, Volume3d, Multipoles3d, ConformerRMSD3D, PharmacophoreFeatures3d, MMFF94 Partial Charges 3d, MMFF94 Energy 3d, Conformer ID 3d, Shape Selfoverlap 3d, Feature Selfoverlap 3d, Shape Fingerprint 3d, and EffectiveRotorCount3D. At the bottom, there is a blue 'FETCH' button and a small copyright notice 'Copyright © DEDaMI 2021'.

Figure B.11: Pubchem repo page for compounds (3D output)

The screenshot shows a web interface for the PubChem repository. At the top, a blue navigation bar contains a home icon, the text 'DEDaMi', and icons for a folder, a key, and a refresh button. Below the navigation bar, a breadcrumb trail reads 'DEDaMi > Cheminformatics > pubchem'. The main content area features the 'PubChem' logo, where the 'C' is inside a blue hexagon. Below the logo are two rows of toggle switches: the first row has '2D' (selected) and '3D'; the second row has 'Substance' (selected) and 'Compound'. Underneath is an 'Input' section with a radio button for 'sid' (selected) and an empty text input field with an information icon. Below that is a radio button for 'Name'. The 'Output' section has a checked radio button for 'All' and unchecked radio buttons for 'Synonyms', 'Standardized Compound', 'Cids', and 'Aids'. At the bottom of the form is a blue button with a download icon and the text 'FETCH'. The footer contains the text 'Copyright © DEDaMi 2021.'

Figure B.12: Pubchem repo page for substances

DEDaMi > Cheminformatics > pubchem

cid : 5090

molecular_formula : C17H14O4S

molecular_weight : 314.4

canonical_smiles : CS(=O)=O)C1=CC=C(C=C1)C2=C(C(=O)OC2)C3=CC=CC=C3

isomeric_smiles : CS(=O)=O)C1=CC=C(C=C1)C2=C(C(=O)OC2)C3=CC=CC=C3

Iupac_name : 3-(4-methylsulfonylphenyl)-4-phenyl-2H-furan-5-one

InchIkey : RZJQGNCSQAWON-UHFFFAOYSA-N

xlogp : 2.3

exact_mass : 314.0612801

monoisotopic_mass : 314.06128010

tpsa : 68.8

complexity : 556

charge : 0

h_bond_donor_count : 0

h_bond_acceptor_count : 4

rotatable_bond_count : 3

heavy_atom_count : 22

isotope_atom_count : 0

atom_stereo_count : 0

defined_atom_stereo_count : 0

undefined_atom_stereo_count : 0

bond_stereo_count : 0

defined_bond_stereo_count : 0

undefined_bond_stereo_count : 0

covalent_unit_count : 1

volume_3d : 239.4

multipoles_3d : [433.12, 9.36, 3.52, 1.27, 12.59, 1.39, 0.01, 2.48, -0.05, -5.01, -0.01, 0.36, 0.39, -0.03]

pharmacophore_features_3d : [6, '1 3 acceptor', '1 4 acceptor', '1 5 acceptor', '5 2 6 8 9 14 rings', '6 11 17 18 20 21 22 rings', '6 7 10 12 13 15 16 rings']

mmff94_partial_charges_3d : [31, '1 1.2', '10 -0.01', '11 0.03', '12 -0.15', '13 -0.15', '14 0.71', '15 -0.15', '16 -0.15', '17 -0.15', '18 -0.15', '19 0.11', '2 -0.43', '20 -0.15', '21 -0.15', '22 -0.15', '25 0.15', '26 0.15', '27 0.15', '28 0.15', '29 0.15', '3 -0.57', '30 0.15', '34 0.15', '35 0.15', '36 0.15', '4 -0.65', '5 -0.65', '6 -0.17', '7 0.03', '8 -0.01', '9 0.42]

mmff94_energy_3d : 55.2398

conformer_id_3d : 000013E200000001

shape_selfoverlap_3d : 934.9

feature_selfoverlap_3d : 30.466

['10366900 7 17846213324252982388', '10616163 171 18410578392157636294', '107951']

isotope_atom_count : 0

atom_stereo_count : 0

defined_atom_stereo_count : 0

undefined_atom_stereo_count : 0

bond_stereo_count : 0

defined_bond_stereo_count : 0

undefined_bond_stereo_count : 0

covalent_unit_count : 1

volume_3d : 239.4

multipoles_3d : [433.12, 9.36, 3.52, 1.27, 12.59, 1.39, 0.01, 2.48, -0.05, -5.01, -0.01, 0.36, 0.39, -0.03]

pharmacophore_features_3d : [6, '1 3 acceptor', '1 4 acceptor', '1 5 acceptor', '5 2 6 8 9 14 rings', '6 11 17 18 20 21 22 rings', '6 7 10 12 13 15 16 rings']

mmff94_partial_charges_3d : [31, '1 1.2', '10 -0.01', '11 0.03', '12 -0.15', '13 -0.15', '14 0.71', '15 -0.15', '16 -0.15', '17 -0.15', '18 -0.15', '19 0.11', '2 -0.43', '20 -0.15', '21 -0.15', '22 -0.15', '25 0.15', '26 0.15', '27 0.15', '28 0.15', '29 0.15', '3 -0.57', '30 0.15', '34 0.15', '35 0.15', '36 0.15', '4 -0.65', '5 -0.65', '6 -0.17', '7 0.03', '8 -0.01', '9 0.42]

mmff94_energy_3d : 55.2398

conformer_id_3d : 000013E200000001

shape_selfoverlap_3d : 934.9

feature_selfoverlap_3d : 30.466

Figure B.13: Pubchem output cut

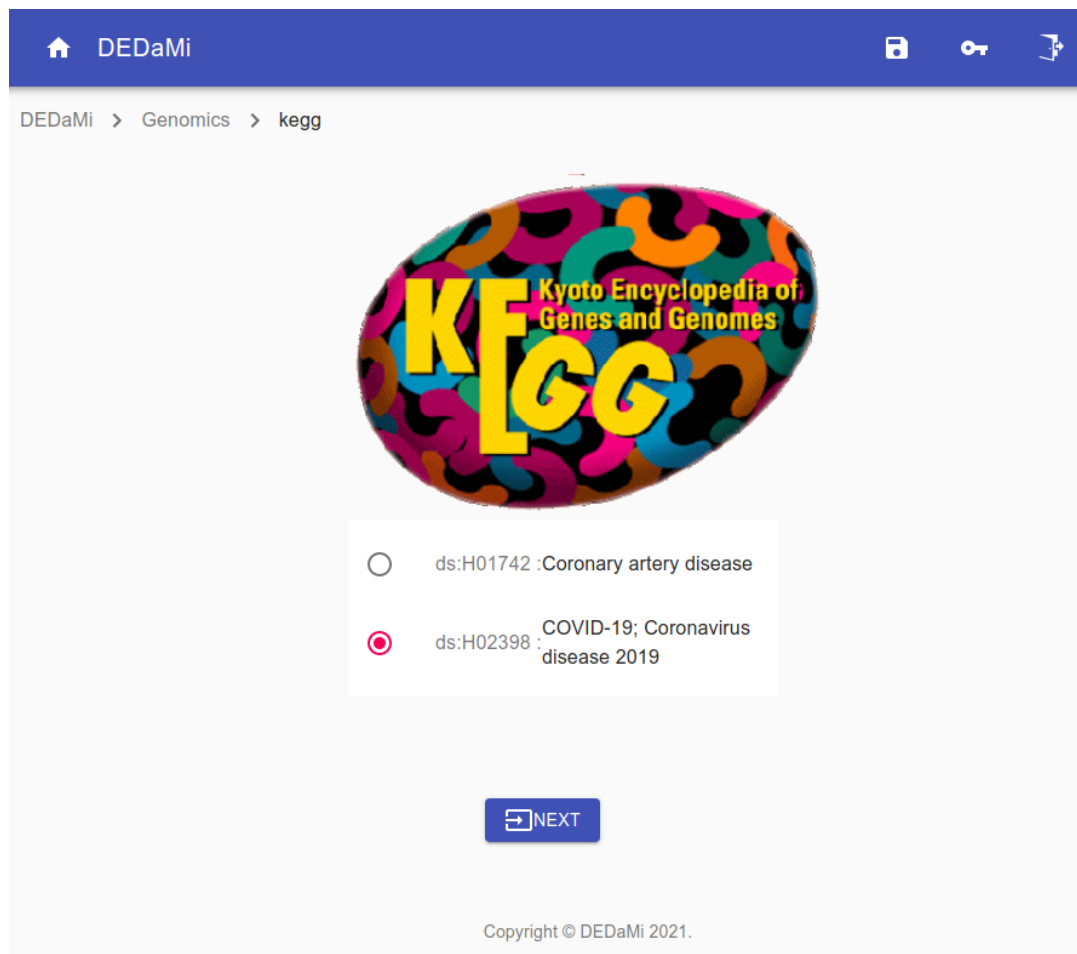


Figure B.14: Kegg search results

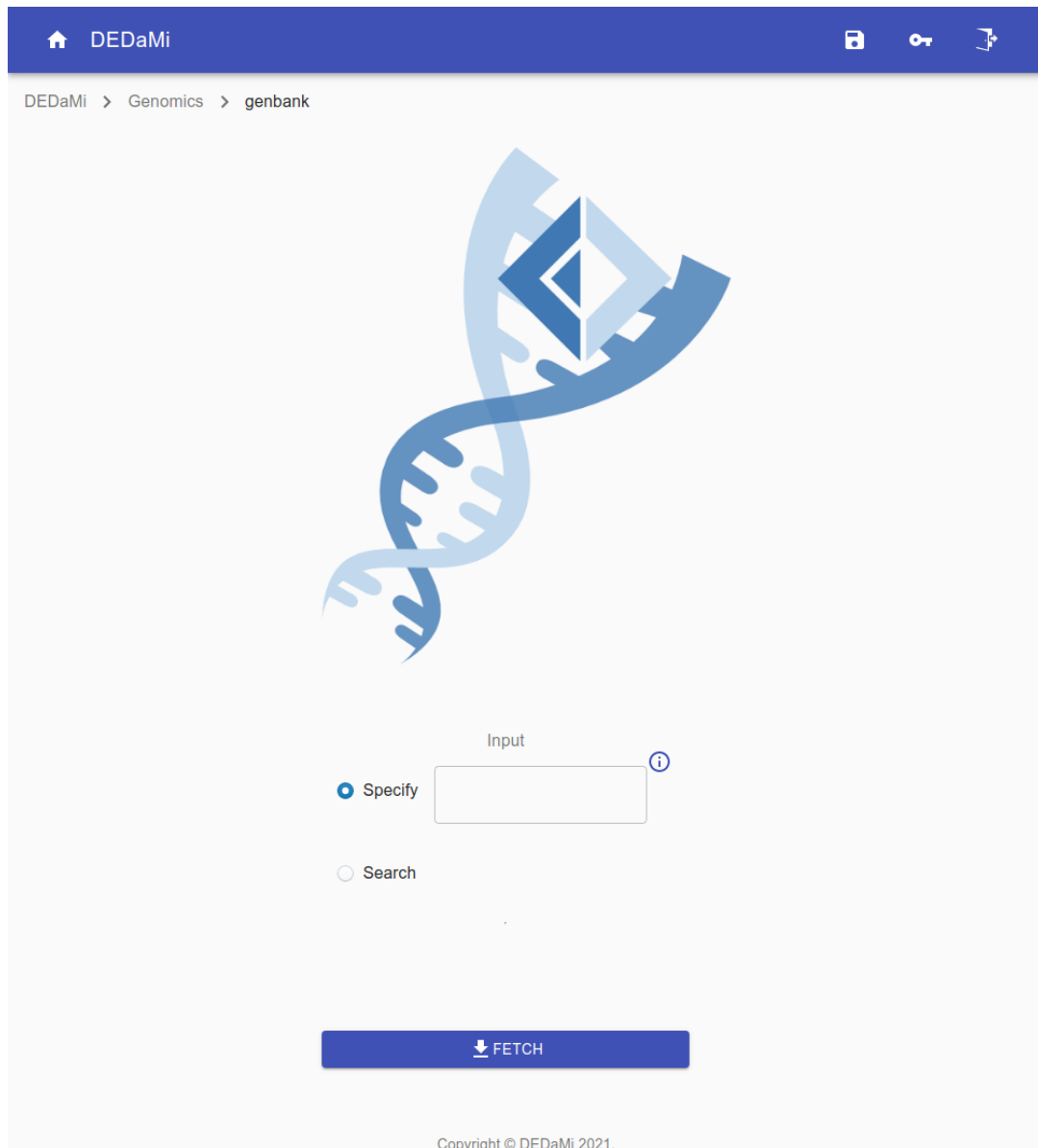


Figure B.15: Genbank repo page

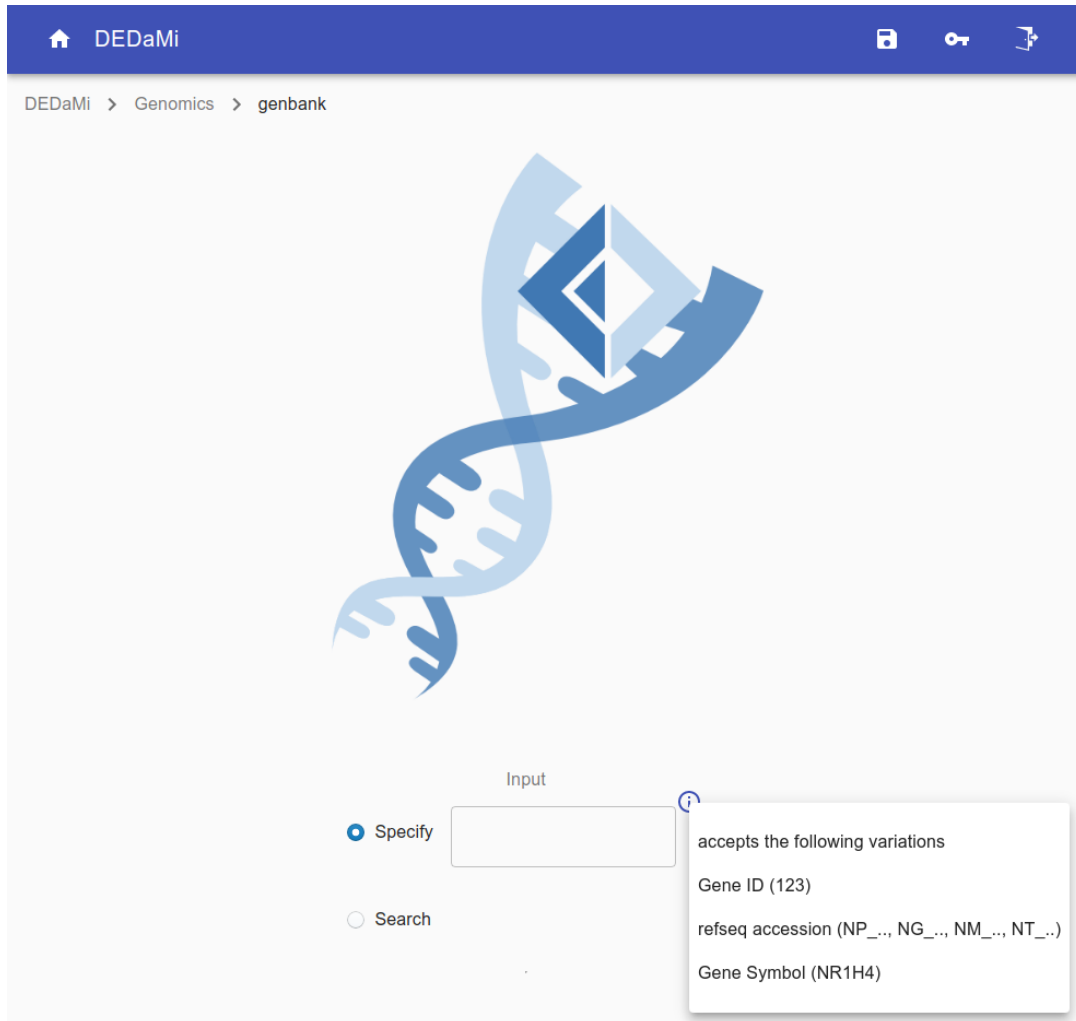


Figure B.16: Genbank repo page with help tooltip

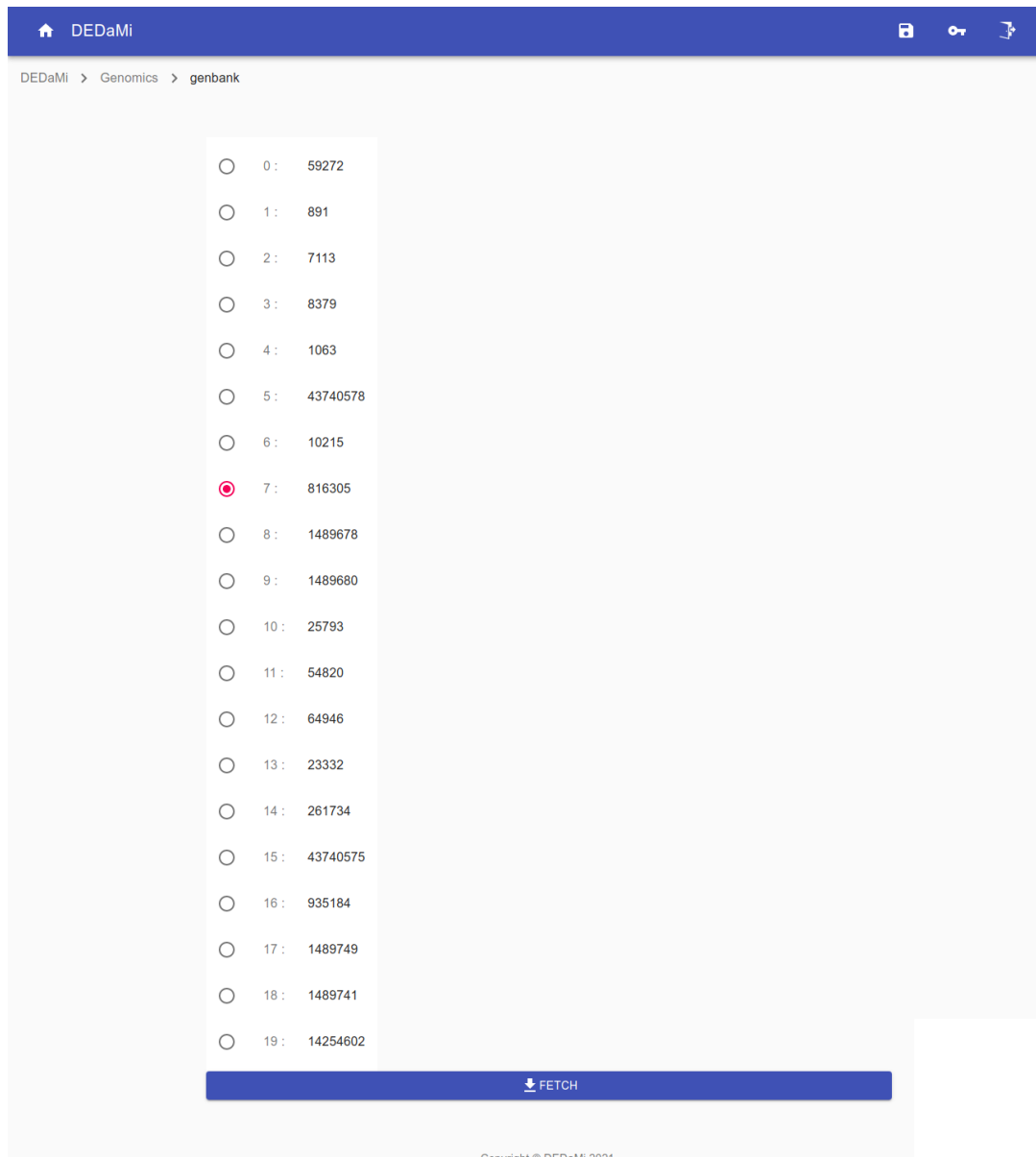
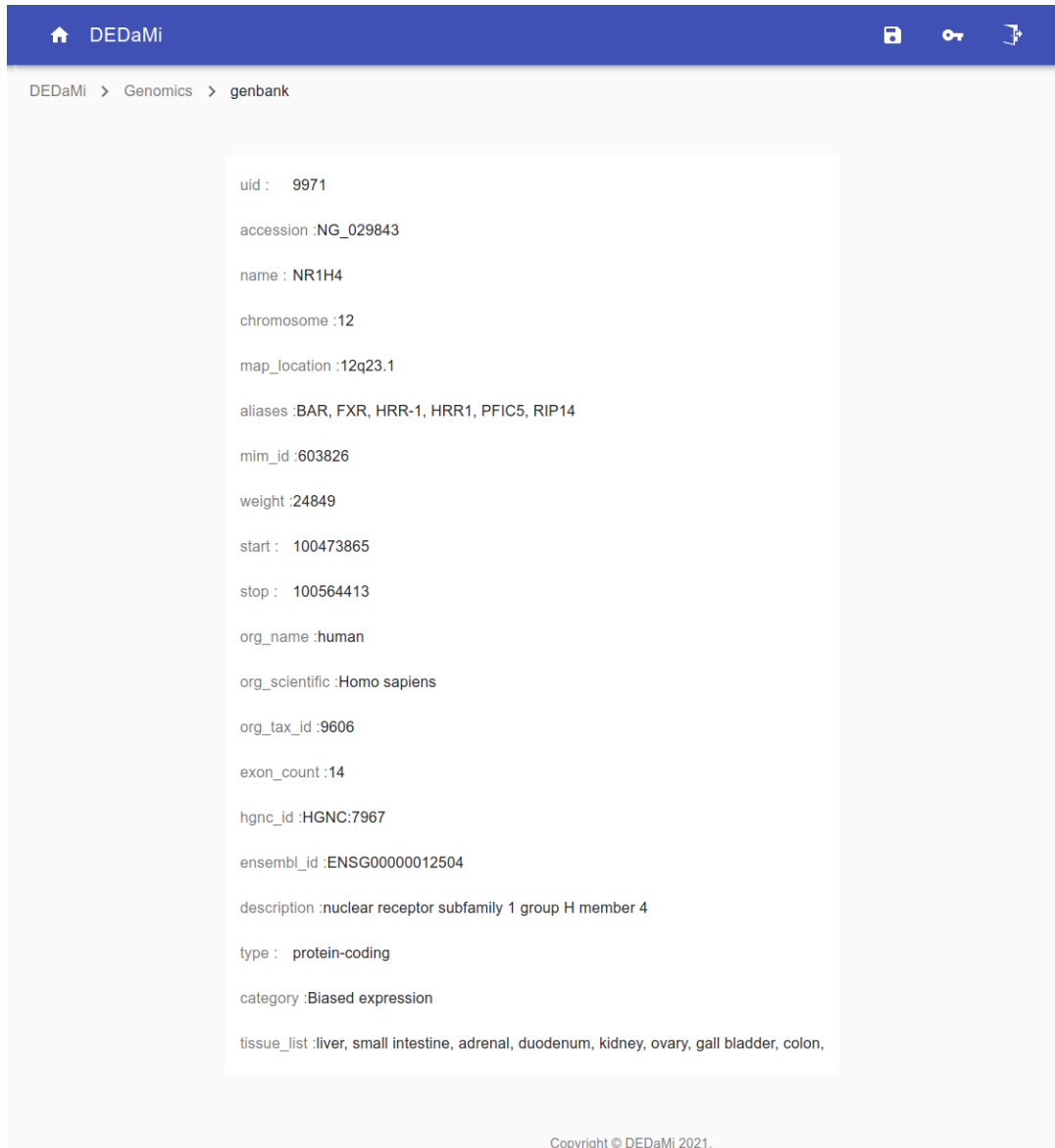


Figure B.17: Genbank search results



The screenshot displays the DEDaMi web interface. At the top, there is a blue header with the DEDaMi logo and navigation icons. Below the header, a breadcrumb trail shows the path: DEDaMi > Genomics > genbank. The main content area features a white box containing the following Genbank-style output:

```
uid : 9971
accession :NG_029843
name : NR1H4
chromosome :12
map_location :12q23.1
aliases :BAR, FXR, HRR-1, HRR1, PFIC5, RIP14
mim_id :603826
weight :24849
start : 100473865
stop : 100564413
org_name :human
org_scientific :Homo sapiens
org_tax_id :9606
exon_count :14
hgnc_id :HGNC:7967
ensembl_id :ENSG00000012504
description :nuclear receptor subfamily 1 group H member 4
type : protein-coding
category :Biased expression
tissue_list :liver, small intestine, adrenal, duodenum, kidney, ovary, gall bladder, colon.
```

At the bottom of the white box, the text "Copyright © DEDaMi 2021." is visible.

Figure B.18: Genbank output page

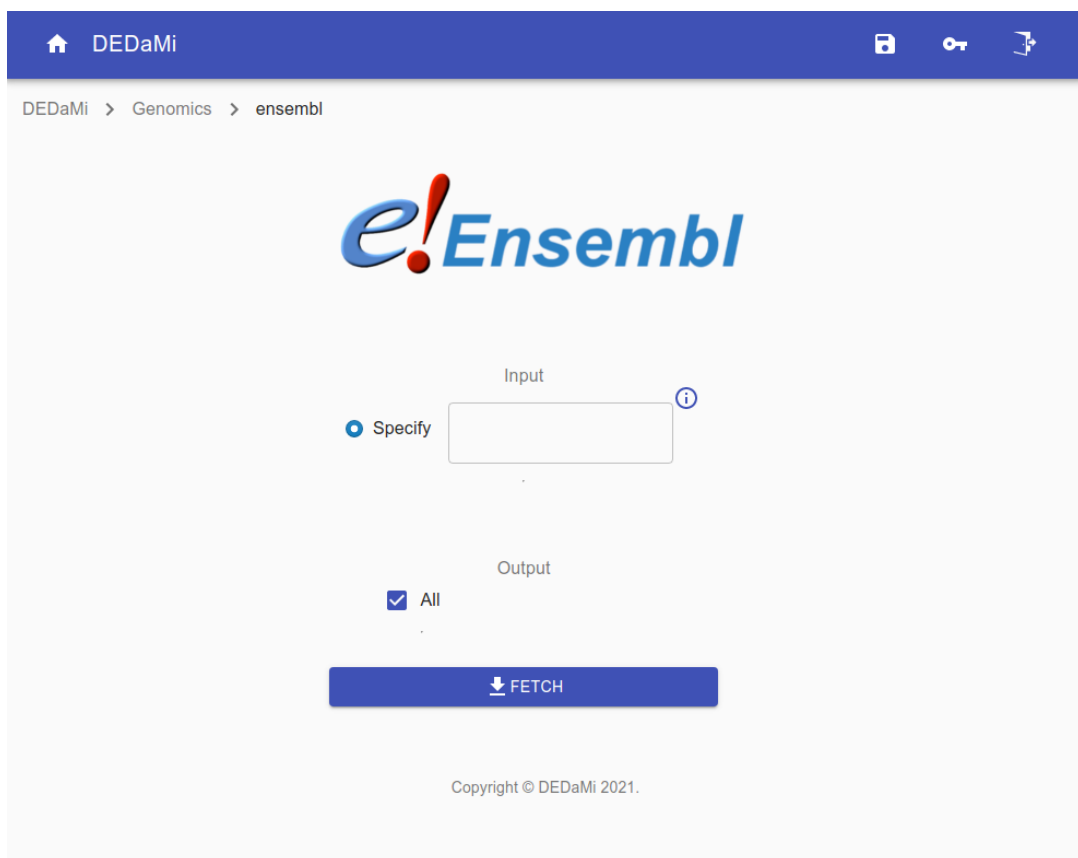
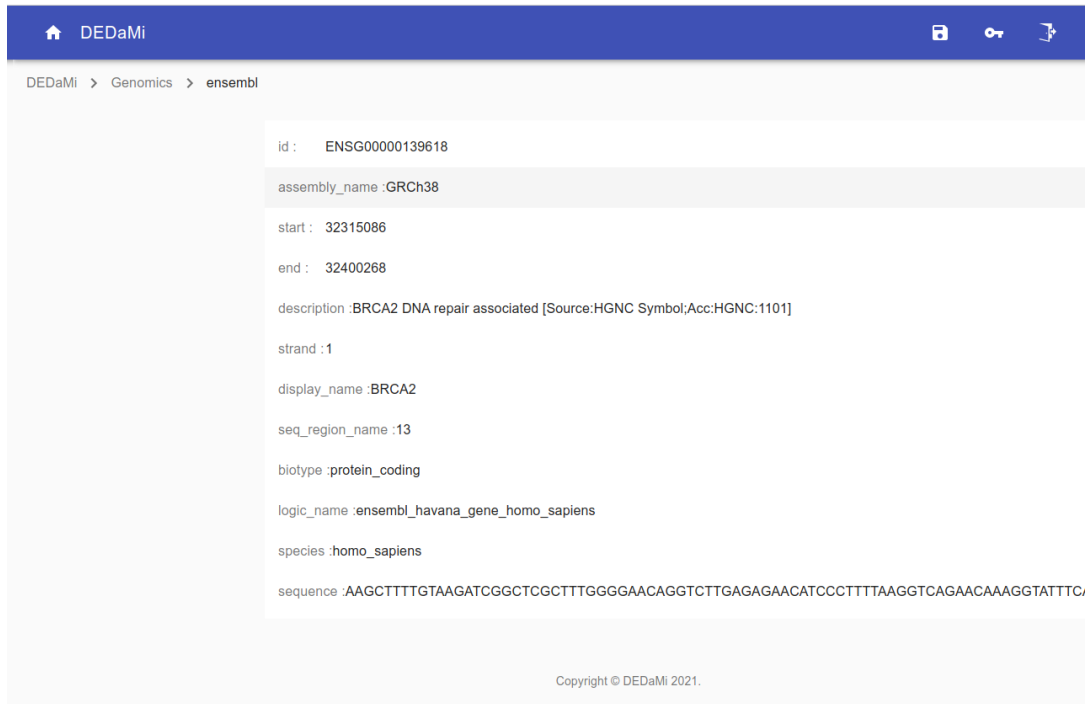


Figure B.19: Ensembl repo page

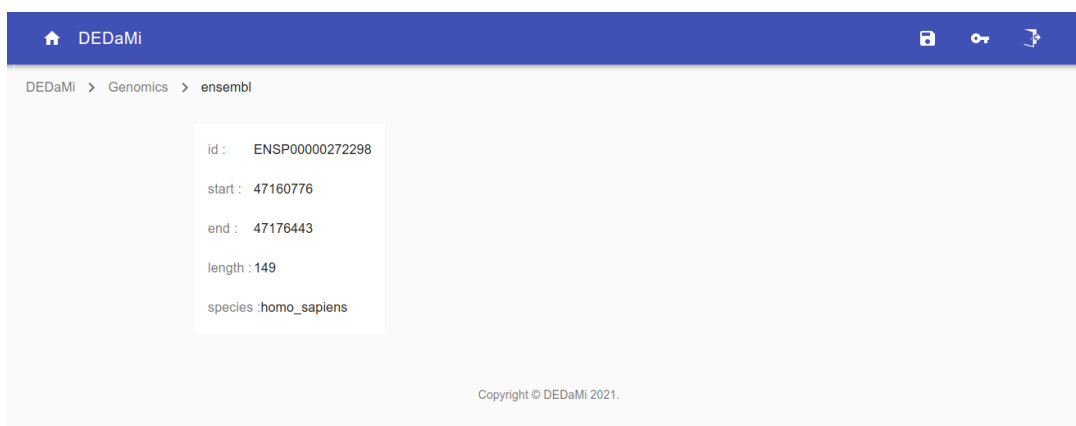


The screenshot shows the DEDaMi web interface. The top navigation bar is blue with a home icon, the text 'DEDaMi', and icons for a folder, a key, and a refresh symbol. Below the navigation bar, the breadcrumb path is 'DEDaMi > Genomics > ensembl'. The main content area displays the following information for the gene ENSG00000139618:

- id : ENSG00000139618
- assembly_name : GRCh38
- start : 32315086
- end : 32400268
- description : BRCA2 DNA repair associated [Source:HGNC Symbol;Acc:HGNC:1101]
- strand : 1
- display_name : BRCA2
- seq_region_name : 13
- biotype : protein_coding
- logic_name : ensembl_havana_gene_homo_sapiens
- species : homo_sapiens
- sequence : AAGCTTTTGTAAAGATCGGCTCGCTTTGGGGAACAGGTCTTGAGAGAACATCCCTTTTAAGGTCAGAACAAAGGTATTTC

At the bottom of the page, there is a copyright notice: 'Copyright © DEDaMi 2021.'

Figure B.20: Ensembl Gene output page



The screenshot shows the DEDaMi web interface. The top navigation bar is blue with a home icon, the text 'DEDaMi', and icons for a folder, a key, and a refresh symbol. Below the navigation bar, the breadcrumb path is 'DEDaMi > Genomics > ensembl'. The main content area displays the following information for the protein ENSP00000272298:

- id : ENSP00000272298
- start : 47160776
- end : 47176443
- length : 149
- species : homo_sapiens

At the bottom of the page, there is a copyright notice: 'Copyright © DEDaMi 2021.'

Figure B.21: Ensembl Protein output page

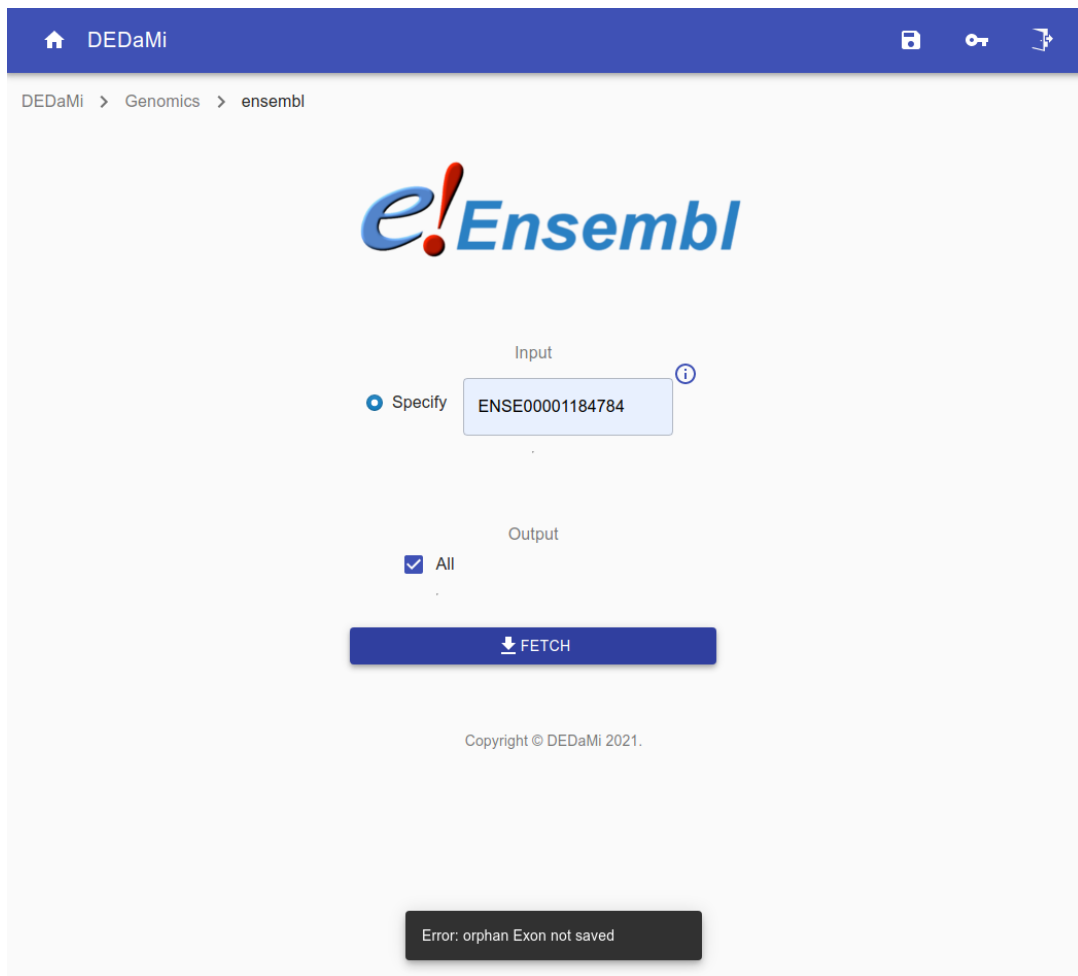



Figure B.22: Orphan Exon not saved

Home DEDaMi



Sign in




User Name *
admin


Password *
.....

SIGN IN

Copyright © DEDaMi 2021.

Figure B.23: Login page

Home DEDaMi   



Update Password

Enter New Password *

Enter Your Password Again *

UPDATE PASSWORD

Copyright © DEDaMi 2021.

Figure B.24: Change Password page

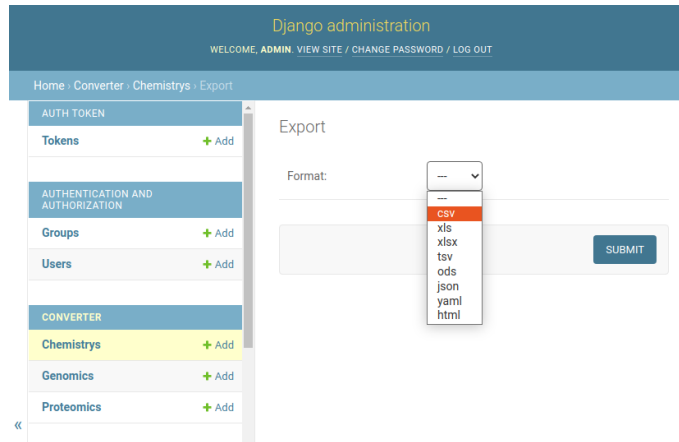


Figure B.25: Database export example

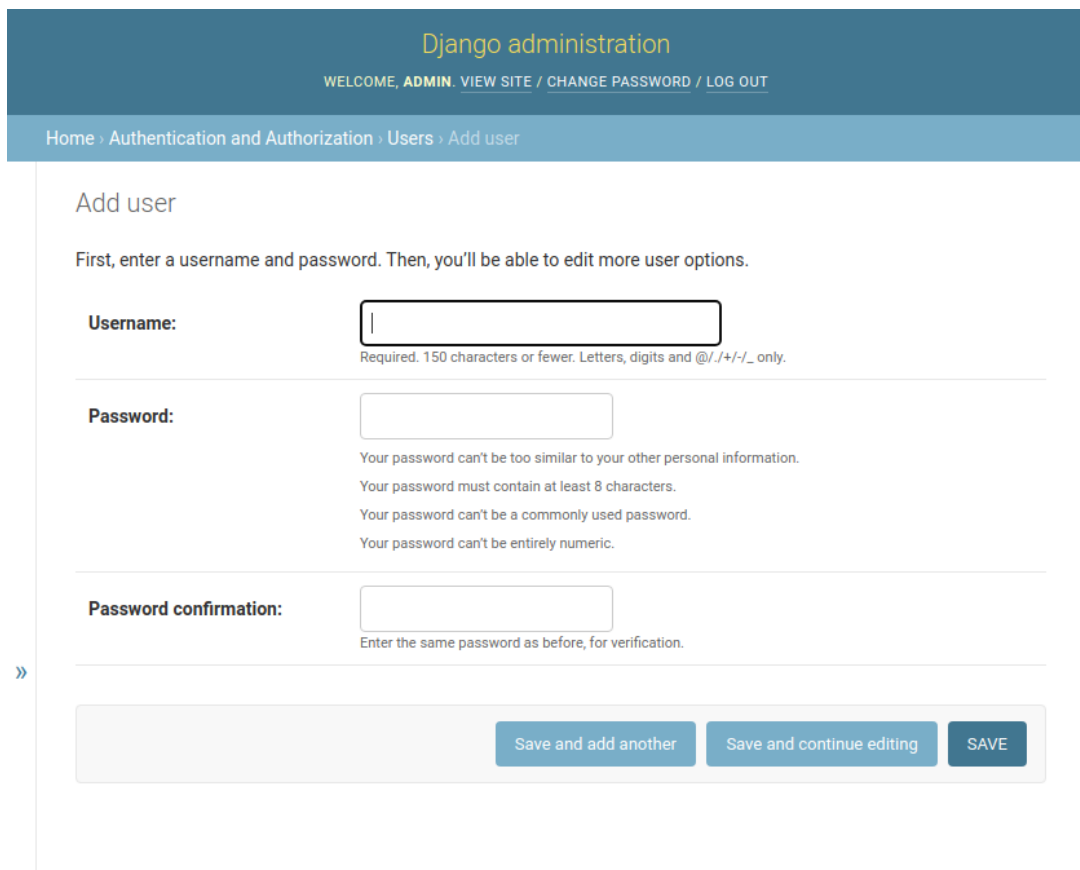


Figure B.26: Add New user page

Django administration

WELCOME, ADMIN | [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Authentication and Authorization > Users > rcamacho

Change user

HISTORY

rcamacho

Username:
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: algorithm: pbkdf2_sha256 iterations: 260000 salt: jHf8NV***** hash: 1sz29X*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:

Last name:

Email address:

Permissions

Active
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Staff status
Designates whether the user can log into this admin site.

Superuser status
Designates that this user has all permissions without explicitly assigning them.

Groups:

Available groups

Choose all

Chosen groups

Remove all

The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

User permissions:

Available user permissions

contenttypes | content type | Can view
 converter | genomic | Can add genom
 converter | genomic | Can change gen
 converter | genomic | Can delete geno
 converter | genomic | Can view genorr
 converter | transcriptomic | Can add tr
 converter | transcriptomic | Can chang
 converter | transcriptomic | Can delete
 converter | transcriptomic | Can view t
 ensembl | exon | Can add exon
 ensembl | exon | Can change exon
 ensembl | exon | Can delete exon

Choose all

Chosen user permissions

converter | chemistry | Can add chemi
 converter | chemistry | Can change ch
 converter | chemistry | Can delete che
 converter | chemistry | Can view cherr
 converter | proteomic | Can add prote
 converter | proteomic | Can change pr
 converter | proteomic | Can delete pro
 converter | proteomic | Can view prote

Remove all

Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

Important dates

Last login: Date: Today | 📅
 Time: Now | 🕒
Note: You are 2 hours ahead of server time.

Date joined: Date: Today | 📅
 Time: Now | 🕒
Note: You are 2 hours ahead of server time.

Delete

Save and add another

Save and continue editing

SAVE

Figure B.27: Edit user page