Scalable Analysis of Large Datasets in Life Sciences

LAEFAQ AHMED

Doctoral Thesis in Computer Science
Stockholm, Sweden 2019
To my respected parents, my beloved wife and my lovely brother and sister
Abstract

We are experiencing a deluge of data in all fields of scientific and business research, particularly in the life sciences, due to the development of better instrumentation and the rapid advancements that have occurred in information technology in recent times. There are major challenges when it comes to handling such large amounts of data. These range from the practicalities of managing these large volumes of data, to understanding the meaning and practical implications of the data.

In this thesis, I present parallel methods to efficiently manage, process, analyse and visualize large sets of data from several life sciences fields at a rapid rate, while building and utilizing various machine learning techniques in a novel way. Most of the work is centred on applying the latest Big Data Analytics frameworks for creating efficient virtual screening strategies while working with large datasets. Virtual screening is a method in cheminformatics used for Drug discovery by searching large libraries of molecule structures. I also present a method for the analysis of large Electroencephalography data in real time. Electroencephalography is one of the main techniques used to measure the brain electrical activity.

First, I evaluate the suitability of Spark, a parallel framework for large datasets, for performing parallel ligand-based virtual screening. As a case study, I classify molecular library using prebuilt classification models to filter out the active molecules. I also demonstrate a strategy to create cloud-ready pipelines for structure-based virtual screening. The major advantages of this strategy are increased productivity and high throughput. In this work, I show that Spark can be applied to virtual screening, and that it is, in general, an appropriate solution for large-scale parallel pipelining. Moreover, I illustrate how Big Data analytics are valuable in working with life sciences datasets.

Secondly, I present a method to further reduce the overall time of the structure-based virtual screening strategy using machine learning and a conformal-prediction-based iterative modelling strategy. The idea is to only dock those molecules that have a better than average chance of being an inhibitor when searching for molecules that could potentially be used as drugs. Using machine learning models from this work, I built a web service to predict the target profile of multiple compounds against ready-made models for a list of targets where 3D structures are available. These target predictions can be used to understand off-target effects, for example in the early stages of drug discovery projects.

Thirdly, I present a method to detect seizures in long term Electroencephalography readings – this method works in real time taking the ongoing readings in as live data streams. The method involves tackling the challenges of real-time decision-making, storing large datasets in memory and updating the prediction model with newly produced data at a rapid rate. The resulting algorithm not only classifies seizures in real time, it also learns the threshold in real time. I also present a new feature “top-k amplitude measure” for classifying which parts of the data correspond to seizures. Furthermore, this feature helps to reduce the amount of data that needs to be processed in the subsequent steps.
Sammanfattning

Vi upplever just nu en flodvåg av data inom både vetenskaplig forskning och företagsdriven utveckling. Detta gäller framförallt inom livsvetenskap på grund av utveckling av bättre instrument och framsteg inom informationsteknologi under den senaste åren. Det finns dock betydande utmaningar med hanteringen av sådana datamängder som sträcker sig från praktisk hantering av de stora datavolymerna till förståelse av betydelsen och de praktiska implikationerna av dessa data.

I den här avhandlingen presenterar jag metoder för att snabbt och effektivt hantera, behandla, analysera och visualisera stora biovetenskapliga datamängder. Större delen av arbetet är fokuserat på att tillämpa de senaste Big Data ramverken för att på så sätt skapa effektiva verktyg för virtuell screening, vilket är en metod som används för att säkra igenom stora mängder kemiska strukturer för läkemedelsutvecklings. Vidare presenterar jag en metod för analys av stora mängder elektrencefalografidata (EEG) i realtid, vilken är en av de huvudsakliga metoderna för att mäta elektrisk hjärnaktivitet.


Acknowledgments

Throughout the research work for this dissertation and during its completion, I received a great deal of help and assistance. First of all, I would like to specially thank my supervisor, Dr. Erwin Laure, who was always helpful – both in academic, as well as administrative, matters – and who guided me with his useful advice to improve my work.

I was lucky to have Dr. Ake Edlund and Dr. Ola Spjuth as my co-supervisors – they provided valuable advice to direct my research and collaborated with me in formulating research questions and methodologies. I would also like to acknowledge all my co-authors and collaborators, especially Dr. Ola’s research group at Uppsala Bio-medicine Centre, Uppsala University. They were the nicest colleagues and were always helpful, both practically and theoretically.

This work would not have been possible without financial support from my home and host universities, the University of Engineering and Technology, Peshawar, and the KTH Royal Institute of Technology, Stockholm, respectively. I am grateful for their support and for showing confidence in my abilities.

I would also like to thank all my colleagues at the PDC Centre for High Performance Computing, and specially Genet Edmondson, who provided her precious time for proofreading my thesis, and Daniel Ahlin, who translated the abstract to Swedish, and Mohammad Rezaei, who was helpful in daily discussions and routine tasks.

I would also like to thank Dr. Vincent Breton for being the opponent and the committee members Dr. Samuel Coulbourn Flores, Dr. Jessica Lindvall and Dr. Panos Papapetrou for being part of the thesis defence and, for their constructive and encouraging discussion. I would also like to thank the Advance Reviewer Dr. Pawel Herman for his constructive comments that helped me to improve the thesis.
# Contents

**Acknowledgments** ix  
**Contents** x  
**List of Abbreviations** xiv  

## 1 Introduction

1.1 Big Data Analytics and Life Sciences: A Motivation 1  
1.2 Research Questions and Thesis Contributions 4  
1.3 List of Scientific Publications 5  
1.4 Thesis Overview 7  

## 2 Background

2.1 Big Data 9  
  2.1.1 Big Data Characteristics 9  
  2.1.2 Big Data Processing Frameworks 10  
2.2 Apache Spark 12  
  2.2.1 Spark Streaming 14  
2.3 Cloud Computing 15  
  2.3.1 Definition 15  
    2.3.1.1 Essential Characteristics 15  
    2.3.1.2 Service Models 15  
    2.3.1.3 Deployment Models 17  
2.4 Microservices 18  
  2.4.1 Deploying Microservices 18  
    2.4.1.1 Docker Containers 18  
    2.4.1.2 Kubernetes 18  
2.5 Machine Learning and Big Data Analytics 19  
  2.5.1 Types of Learning 19  
    2.5.1.1 Supervised Learning 19  
    2.5.1.2 Unsupervised Learning 20  
  2.5.2 Machine Learning Algorithms 20  
  2.5.3 Conformal Prediction 21
## Contents

### 2.5.3.1 Inductive Conformal Prediction

### 2.5.4 Machine Learning Libraries for Big Data Analytics

### 2.6 Cheminformatics

#### 2.6.1 Drug Discovery

#### 2.6.2 Virtual Screening

##### 2.6.2.1 Ligand-Based Virtual Screening

##### 2.6.2.2 Structure-Based Virtual Screening

##### 2.6.2.3 Off-Target Predictions

### 2.7 Neuroinformatics

#### 2.7.1 Electroencephalography

### 2.8 Summary

### 3 Related Work

#### 3.1 Virtual Screening

##### 3.1.1 Parallel and Distributed Methods

##### 3.1.2 Intelligent Iterative Methods

##### 3.1.3 Predicted Target Profiles

#### 3.2 Electroencephalography

#### 3.3 Summary

### 4 Parallel Modelling For Data Intensive Virtual Screening

#### 4.1 Introduction

#### 4.2 Using Iterative MapReduce for Parallel Virtual Screening: Feasibility Study for Spark

##### 4.2.1 Virtual Screening and LibSVM

##### 4.2.2 Parallel Ligand-Based Virtual Screening Implementation

##### 4.2.2.1 Tools and Infrastructure

##### 4.2.2.2 Architecture of Parallel Ligand-Based Virtual Screening

##### 4.2.3 Workflow of the Driver Program

#### 4.3 Large-scale Structure-Based Parallel Virtual Screening on public cloud resources with Apache Spark

##### 4.3.1 SBVS Pipelines

##### 4.3.1.1 Data Input

##### 4.3.1.2 Parallel Screening

##### 4.3.2 Experiments

##### 4.3.2.1 Environment Settings

##### 4.3.2.2 Benchmarking

##### 4.3.2.3 Scaling and Speed-Up: Performance Evaluation
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALS</td>
<td>Alternating Least Squares</td>
</tr>
<tr>
<td>Amp&lt;sub&gt;top-k&lt;/sub&gt;</td>
<td>Average top-k Amplitude</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>BAN</td>
<td>Body Area Network</td>
</tr>
<tr>
<td>BCI</td>
<td>Brain Computer Interface</td>
</tr>
<tr>
<td>CDK</td>
<td>Chemistry Development Kit</td>
</tr>
<tr>
<td>CMA</td>
<td>Cumulative Mean Amplitude</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CPVS</td>
<td>Conformal Prediction Based Virtual Screening</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute Unified Device Architecture</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>DNA</td>
<td>Deoxyribonucleic Acid</td>
</tr>
<tr>
<td>DOVIS</td>
<td>Docking-based Virtual Screening</td>
</tr>
<tr>
<td>DUD</td>
<td>Directory of Useful Decoys</td>
</tr>
<tr>
<td>EEG</td>
<td>Electroencephalography</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract, Transform, Load</td>
</tr>
<tr>
<td>GBSA</td>
<td>Generalized Born Surface Area</td>
</tr>
<tr>
<td>GFS</td>
<td>Google File System</td>
</tr>
<tr>
<td>GMR</td>
<td>Google MapReduce</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>GPGPU</td>
<td>General Purpose Computing on GPU</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>HTS</td>
<td>High-Throughput Screening</td>
</tr>
<tr>
<td>IAAS</td>
<td>Infrastructure As A Service</td>
</tr>
<tr>
<td>ICA</td>
<td>Independent Component Analysis</td>
</tr>
<tr>
<td>ICP</td>
<td>Inductive Conformal Prediction</td>
</tr>
<tr>
<td>INCF</td>
<td>International Neuroinformatics Coordinating Facility</td>
</tr>
<tr>
<td>LBFGS</td>
<td>Limited Memory Broyden-Fletcher-Goldfarb-Shanno Algorithm</td>
</tr>
<tr>
<td>LBVS</td>
<td>Ligand-Based Virtual Screening</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>MR</td>
<td>MapReduce</td>
</tr>
<tr>
<td>NCE</td>
<td>Novel Chemical Entity</td>
</tr>
<tr>
<td>PAAS</td>
<td>Platform As A Service</td>
</tr>
<tr>
<td>PEs</td>
<td>Processing Elements</td>
</tr>
<tr>
<td>PDB</td>
<td>Protein Data Bank Archive</td>
</tr>
<tr>
<td>PTPAAS</td>
<td>Predicting Target Profiles As A Service</td>
</tr>
<tr>
<td>PVM</td>
<td>Parallel Virtual Machine</td>
</tr>
<tr>
<td>RDD</td>
<td>Resilient Distributed Datasets</td>
</tr>
<tr>
<td>SAAS</td>
<td>Software As A Service</td>
</tr>
<tr>
<td>SAR</td>
<td>Structure-Activity Relationship</td>
</tr>
<tr>
<td>SBVS</td>
<td>Structure Based Virtual Screening</td>
</tr>
<tr>
<td>SDF</td>
<td>Standard File Format</td>
</tr>
<tr>
<td>SNN</td>
<td>Spiking Neural Network</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSC</td>
<td>Snic Science Cloud</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machines</td>
</tr>
<tr>
<td>TCP</td>
<td>Transductive Conformal Prediction</td>
</tr>
<tr>
<td>QSAR</td>
<td>Quantitative Structure-Activity Relationship</td>
</tr>
<tr>
<td>WSE</td>
<td>Weak Scaling Efficiency</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Big Data Analytics and Life Sciences: A Motivation

In the past decade, academic researchers and businesses have reached the point where they possess the capacity to produce massive amounts of data – this is because of the major developments that have occurred both in the field of information technology and in the capabilities of scientific equipment. The massive sets of data that are now routinely being produced are commonly referred to as Big Data. In order to deal with these large datasets, new technologies – known as Big Data Analytics – are being developed. These technologies are becoming prevalent, and have the advantage that they provide better insight into the inherent meaning of these vast amounts of data in less time and at a cheaper cost. This, in turn, improves the decision-making processes for people who depend on the data. Big Data Analytics also make it possible to direct the relevant parts of the data (or the relevant portions of results from computations using the data) to the different groups of people who are only interested in using certain parts of the data – this is known as providing finer end user segregation, and saves time as users do not have to sift through the data to isolate the information that they need. In addition, Big Data Analytics make it easy to launch new and innovative tailor-made services for the end users of the data.

Big data is now prevalent in many different fields of the life sciences, for example as genomic data for DNA sequencing in the area of biomedical computing [157], in chemical libraries for drug discovery in the field of cheminformatics, as continuous neural data in neuroinformatics, and in patient historic health data in the health sciences. Just to give an idea of the large size of the datasets generated in these types of projects, the 1000 Genomes Project has produced more than 100 Terabytes(TB) [15] of data, and it is predicted that the 100K Genome project [1] will generate approximately 100 times more data. It is widely believed that the use of Big Data Analytics will enable us to make better decisions, and bring significant benefits (such as cheaper health care [32], appropriate personalised medical treatment [109]...
and the prevention of drug related side effects [155].

Today’s researchers are privileged to have extremely large sets of data to study in order to obtain a better understanding of the problems we face. The increasing volume, high dimensionality and structure of this data create novel challenges for researchers when it comes to processing and analysing the data [97]. High dimensional data (high number of parameters in data) raises the challenges of noise aggregation e.g., it has been shown [95, 110] that due to noise aggregation, in higher dimensions, the classification rules performs no better than random guess. High dimensional data also creates false correlations between the response variable and the irrelevant parameters that can result in false inferences and wrong scientific conclusions [96]. High dimensionality combined with massive volumes of data gives rise to problems such as huge computational costs and algorithmic unpredictability. Other challenges that can arise are related to the diversity of the data, experimental variations and statistical biases that are caused by the variations in data that comes from different sources and in different forms (such as text, images, audio, or video).

As the size of sets of data increases, more and more computational resources are required for storing and analysing the data. Furthermore, to support the storage and analysis of the data, a number of databases and software applications are needed [115]. Large volumes of data also give rise to other issues, the most common being scalability. (An algorithm is said to be scalable if it keeps performing well as the size of the system that it is applied to increases. As the volume of big data tends to keep growing, it is important to use scalable algorithms to handle the data.) Big datasets tend to display some general characteristics, such as the volume, velocity, veracity, variety and value of the data [183], and each of these characteristics can cause challenges in their own right. (The characteristics of big data are discussed further in Chapter 2.) With the recent developments in the life sciences and the consequent availability of many problems with extremely large datasets, the life sciences have become a data science field. Thus life sciences researchers face the same challenges as those that arise from the famous big data “v”s (i.e. volume, variety, velocity). Ultimately though the most important question is how we get value from big data by solving these challenges.

Often, massive datasets are produced continuously, and at a very fast rate, in real time e.g. the New York Stock Exchange captures 1 TB of trade data during each trading session, and hundreds of car sensors produce data continuously resulting in TBs of data being produced each hour. Such data needs to be processed in real time so that decisions, which depend on the data, can be made in real time (rather than requiring long periods of processing before decisions are made). The challenges that must be overcome to make this possible are quickly transferring data from where it is produced to where it will be processed (and without losing data during the transfer), analysing the data in real time while updating the algorithm in real time to incorporate all the newly produced data, and efficiently storing large quantities of data in limited memory for faster access. Applications exist in the life sciences where we need to process the data as soon as it is generated and provide results.
in real time e.g. the body area network (BAN) is a collection of wearable devices to monitor the health of a patient. These wearable devices usually consist of many sensors that collect current health data for the patient e.g. heart rate and body temperature. Combining and analysing such data in real time, when it is being generated at a rapid rate, is a tedious task and cannot be performed manually.

Large quantities of data that is heterogeneous and unstructured cannot be efficiently manipulated by conventional relational databases. Conventional data mining techniques, which have been successfully used by researchers previously, are not good for tackling large datasets either. Similarly, in the life sciences, the processes that are being investigated are very complicated due to their nature, and therefore the resulting datasets are also inherently complex, diverse and unstructured [161]. This aspect of the nature of life sciences data acts as the greatest obstacle to us obtaining value from the data, for example by mining new knowledge and making scientific discoveries.

To this end, different parallel programming frameworks have been implemented that enable us to solve these big data problems in parallel – such as MapReduce [81], Google File System [105], Big Table [74], Dremel [159], Pregel [154], and their open-source alternatives like Hadoop Ecosystem [61], Mesos [117] and Spark [235]. These frameworks have been widely and successfully adopted in business, however, these tools have not yet been adopted by the eScience research communities and hence the applicability of these tools in life sciences problems is not yet completely understood.

Such frameworks also include parallel machine learning (ML) algorithm implementations to bring the hidden value out of big data, which has not been possible to date with conventional data mining techniques. ML algorithms have been available for decades but most of them were implemented to work only in serial. Even if some of the algorithms were developed for parallel use, such large datasets have never been available previously and consequently the issues related to large datasets were unknown. Therefore, it is necessary to either build new algorithms from scratch that can handle large datasets, or to use algorithms that are already available but with novel strategies to make it possible to solve big data problems.

In this thesis, I present standards, algorithms, novel strategies and software implementations for several large dataset problems in the field of life sciences. Most of the work is centred on applying the latest Big Data Analytics frameworks for creating efficient virtual screening strategies for large datasets. The work also includes the development of a web service for creating Predicted Target profiles of multiple compounds against a list of targets. A method for analysing large quantities of Electroencephalography (EEG) data in real time was also implemented.

On the basis of the challenges that have been discussed and the potential of Big Data Analytics frameworks, the motivation of this thesis is to fulfil the following goals.

- Investigating the ongoing research in the area of eScience adoption of next-generation Big Data Analytics methods
CHAPTER 1. INTRODUCTION

• Increasing the efficiency and usefulness of the existing programming models to meet the needs of life sciences researchers

• Investigating the applicability and scalability performance of Spark, a next-generation Big Data Analytics framework, when applied to large-scale life sciences problems

• Adapting previously well-known techniques for large dataset problems using Spark

1.2 Research Questions and Thesis Contributions

The main aim of the thesis is to create efficient methodologies for analysing large datasets in life sciences problems using Spark in a scalable manner on commodity computing resources. This is important because it enables life sciences researchers to derive knowledge from large datasets with ease and much more rapidly, thus reducing the time and costs involved in the processes. Today, the majority of the life sciences problems use high performance computing (HPC). This research demonstrates that Big Data Analytics for some problems are useful, has advantages, or is an alternative to HPC. It is important because small and medium-sized organizations often cannot afford expensive HPC resources of their own so they can instead use these methodologies on commodity computing resources. The major research questions (Q1, Q2 and Q3) addressed in this thesis are outlined here, along with the significant contributions (C1, C2 and C3) that are the basis of the thesis.

• Q1: How can the next-generation Big Data Analytics methods be adopted in life sciences problems and provide ready-made solution for large-scale problems with better resource utilization and speed-up?

• Q2: How can the efficiency of Big Data Analytics methods in life sciences be further increased using machine learning methods and how can intelligent end user tools based on them be provided?

• Q3: How can the efficiency of real-time stream analysis in life sciences problems be improved in an online learning setting?

• C1: Our first contribution of the thesis is the evaluation of Spark as a scalable tool for virtual screening, a technique in cheminformatics used for Drug discovery by searching large libraries of molecule structures and decreasing the overall execution time through data parallelism. In the first part of the contribution, we performed ligand-based virtual screening (LBVS) i.e. using Quantitative Structure-Activity Relationship (QSAR) models with molecular signatures and SVM. We showed how Spark and HDFS could be used in combination with ML algorithms e.g. to distribute and analyse data with SVM in parallel, providing a satisfactory scaling behaviour and efficiency. While
traditionally SVM based applications are implemented in MPI, we showed that MapReduce is a viable alternative that opens the path for efficient exploitation of large-scale public cloud infrastructures for virtual screening. In the second part of the contribution, we performed large-scale structure based virtual screening (SBVS) i.e. with molecular docking. We also created a new parallel virtual screening pipeline that provides a ready-made tool to the cheminformatics community to perform large-scale virtual screening efficiently (chapter 4). Previously these tools have not been used by the cheminformatics community and as such, with this work, we showed a new faster scalable method for virtual screening to the community. The resulting publications from this contribution are [38, 153].

- **C2**: The second contribution is a new conformal prediction based, a further improved, iterative virtual screening method that provides the results with similar accuracy to previous methods but in a much shorter time. The docking step in virtual screening is time consuming due to the large size of molecule libraries. Conformal prediction along with SVM allows us to confidently filter out the poor candidates and thus only lead-like molecules are docked. On top of this, we also provided a web service for computing target profiles for Novel Chemical Entities (NCE), docking against a battery of targets where 3D Structures are available. The novel part in developing these target profiles is that they are based on docking scores of molecules docked rather than the ligand-target binding values from the binding databases. This helps life scientists to compute structure-based target predictions. These target predictions can be used to predict off-target effects, for example in early stages in drug discovery projects. The service is implemented in a web application framework play2.0 and deployed as a microservice on OpenShift Origin (chapter 5). The outcome publications from this contribution are [40, 37].

- **C3**: The third contribution is a parallel lightweight method for epileptic seizure detection in large EEG data as real-time streams. In this contribution, we have provided methods for solving challenges of fitting large datasets in memory and quickly adding the newly produced data to the online predictive model without any data loss. In an experimental scenario, our lightweight algorithm was able to detect seizures in real time with low latency and with good overall seizure detection rate. Also, we have introduced a new feature, “top-k amplitude measure” for data reduction. On the basis of the results, we believe this method can be useful in clinics for epileptic patients (chapter 6). The resulting publication from this work is [39].

### 1.3 List of Scientific Publications

This thesis is based on the following publications; the papers are referred in the body of the thesis by their Roman numerals.
CHAPTER 1. INTRODUCTION

Paper I: Using Iterative MapReduce for Parallel Virtual Screening
Laeeq Ahmed, Ake Edlund, Erwin Laure and Ola Spjuth
In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on (Vol. 2, pp. 27-32)

Author’s contributions: I am the main author. I partially contributed to developing the concept underlying the paper and I contributed fully to implementing it.

Paper II: Large-scale virtual screening on public cloud resources with Apache Spark
Marco Capuccini, Laeeq Ahmed, Wesley Schaal, Erwin Laure and Ola Spjuth
Journal of cheminformatics, 9(1), 15.

Author’s contributions: I am the second author. I partially contributed to developing the concept underlying the paper and I contributed partially to implementing it.

Paper III: Efficient iterative virtual screening with Apache Spark and conformal prediction
Laeeq Ahmed, Valentin Georgiev, Marco Capuccini, Salman Toor, Wesley Schaal, Erwin Laure and Ola Spjuth
Journal of cheminformatics, 10(1), 8.

Author’s contributions: I am the main author. I partially contributed to developing the concept underlying the paper and I contributed fully to implementing it.

Paper IV: Predicting Target Profiles with Confidence as a Service using Docking Scores
Laeeq Ahmed, Hiba Alogheli, Staffan Arvidsson, Arvid Berg, Jonathan Alvarsson, Wesley Schaal, Erwin Laure and Ola Spjuth
Manuscript sent to peer reviewed journal

Author’s contributions: I am the main author. I partially contributed to developing the concept underlying the paper and I contributed fully to implementing it.

Paper V: Parallel Real Time Seizure Detection in Large EEG Data
Laeeq Ahmed, Ake Edlund, Erwin Laure, and Stephen Whitmarsh
1.4 Thesis Overview

Figure 1.1 gives an overview of the thesis. In Paper I, I performed a ligand-based virtual screening to filter out molecules using machine learning models in a scalable
manner using Spark. In Paper II, I presented a structure-based virtual screening method that involves molecular docking in a parallel manner using Spark. Adding on to these two parallel methods, I presented an iterative conformal-prediction based virtual screening strategy in Paper III to make the process more efficient by docking and building machine learning models iteratively. In Paper IV, I utilized these machine learning models and presented a web service for predicting target profiles by deploying multiple models in a containerized environment on top of OpenShift origin, an open-source Kubernetes distribution. In Paper V, I presented a study for seizure detection in real-time large EEG data in a streaming manner using Kafka and Spark.

Chapter 2 gives a detailed overview of the theory and the technologies used in the thesis. Chapter 3 summarises relevant work that has been done by other researchers in the area of Big Data Analytics in the life sciences. Chapter 4 provides details about Paper I and Paper II. Chapter 5 covers Paper III and Paper IV. Chapter 6 contains details about Paper V. Chapter 7 discusses the impact and implications of scalable analysis of large datasets in life sciences from a broader perspective. Finally, Chapter 8 presents the conclusions from the research in this thesis and proposes future work to extend this research.
Chapter 2

Background

2.1 Big Data

Over the last two decades, we have seen increasing amounts of data being produced in almost all areas of business and academic research. The data comes from many different sources, such as hundreds of sensors attached to commodity electronic devices and wearable devices, social media, the financial sector, health records, online transactions, search queries, click streams, and scientific machinery [183]. The resulting large datasets, which have not been available previously, open up new insights into their associated domains. Now businesses can make better decisions or offer precise personalized deals, and researchers can make more reliable predictions.

The potential of big data can be measured by the investments made by governments and major technological companies. In 2012, the US government announced a 200 million dollar big data initiative [226]. Recently, Walmart has announced that they will be building the largest private cloud to process 2.5 petabytes of data every hour [51]. However, it is important to remember that the benefits which can be derived from the information in these large datasets can only be realised when we properly understand both the characteristics of such data and the challenges related to these characteristics.

2.1.1 Big Data Characteristics

Big data has five main characteristics known as the five “v”s of big data.

- **Volume** refers to the huge size of the big datasets that are being produced every moment. These datasets are now measured in zetabytes, or even yottabytes, where previously data volumes were typically being measured in gigabytes. Internet companies are playing a major part in producing this data, e.g. Twitter published more than 500 million tweets per day in August 2013 and currently 6000 tweets are published every second [29].
• **Velocity** refers to the rate at which these large datasets are being produced and the speed at which this data has to be processed in order to make real-time decisions, e.g. trading systems can analyse the sentiments expressed in social media data to give indications to either buy or sell shares [98].

• **Veracity** refers to the level of trustworthiness and appropriateness of the information in the big datasets. Tweets that contain typing mistakes and hash tags are one example of data with a low level of veracity. As another example, in 2016, Microsoft had to take their Tay.io chatbot [141] down as it had been trained on Twitter tweets that transformed it from an innocent chatbot to one expressing racist opinions.

• **Variety** means that different types of data are being produced and included in the big datasets. The types of data range from metadata-like XML files to images and video files. The datasets that include these different types of data are consequently complex and unstructured in nature. Previously datasets used to be structured e.g. in sales management systems, data used to be stored in relational databases. Since big datasets are unstructured in nature, it is not easy to process and store the datasets, and hence new technologies are needed.

• **Value** is the most important characteristic of big data and refers to the benefits arising from the discoveries that can be made by analysing these large datasets. Big data is changing the paradigm of information management by giving new insights, and demonstrating new patterns detected from the data, and thus providing value to businesses and researchers, as well as people in other walks of life.

### 2.1.2 Big Data Processing Frameworks

Based on the characteristics discussed in Section 2.1.1, big data is classified into different categories, each of which has specific programming requirements associated with it. The following are the big data processing framework categories and the most well known technologies for handling each of the categories of data.

• **Batch Processing Systems** - Batch processing is the most general form of big data processing. Google is considered to be the pioneer of big data technologies. They presented their famous Google MapReduce (GMR) paper [81] in 2005. The idea was to have two functions, Map and Reduce. Operations like filtering/sorting could be performed with the Map function, while Reduce would be used to summarize the output from Map. The Map and Reduce functions were already present in functional programming languages but had never been used before with large datasets in a distributed environment on commodity servers. Google, at that time, had rather straightforward problems – in particular they needed to search for specific data in crawled
documents. The major challenge they faced was writing complex code for the parallelization of the computations, especially for handling failures. The GMR approach not only solved Google’s basic problems, such as searching for data, but it also made parallelization, fault-tolerance, data distribution and load balancing transparent.

Hadoop is an open-source implementation of GMR, implemented in Java, which provides similar functionality to GMR, i.e. distributed processing of large datasets on commodity servers with scalability, high availability and automatic fault tolerance.

GMR and Hadoop, being distributed processing systems, need to have data stored in a distributed manner. The Google File system (GFS) and Hadoop distributed file system (HDFS) are distributed file systems used by GMR and Hadoop respectively. Other than storing large datasets in a distributed manner, saving on I/O operations is the main advantage they offer. Once the data is stored, it does not need to be moved around and can be processed directly at the location where it is stored by the relevant processing model.

**figure 2.1: Processing Elements in a Stream Processing Engine**

- **Stream Processing Systems** - Stream processing systems process continuous data as soon as it is generated. Examples of applications producing streaming data are patient vital signs monitoring, personal health devices, web traffic processing and forex trading.

Stream is a sequence of events or tuples made available continuously over time in the form \((a_1, a_2, \ldots, a_n, t-1)\) \((a_1, a_2, \ldots, a_n, t)\) \((a_1, a_2, \ldots, a_n, t+1)\), where \(a_i\) represents a characteristic and \(t\) denotes the time.
Stream Processing engine creates a logical network of processing elements (PEs) connected in a Directed Acyclic Graph (DAG) as shown in Figure 2.1. A processing element is a processing unit in a stream processing engine. PEs run in parallel, independently and asynchronously, and communicate between themselves through messages. The output from a PE can be one single tuple, single tuples produced after regular intervals, or nothing at all. Aurora [34], Borealis [33], SPADE [104] and StreamIt [209] are examples of initial event stream processing systems. More recent distributed map-streaming processing systems are Apache Storm [47], Apache S4 [5], Apache Samza [6], Spark Streaming [236] and Twitter’s Heron [140]. Azure Stream Analytics [8], Google MillWheel [44] and Amazon Kinesis [2] are some of the commercially available stream processing systems. Table 2.1 gives a comparison of some of the open-source stream processing systems.

- Hybrid Systems - Hybrid systems provide multiple data analysis facilities in a single stack. Such systems are preferred because the users do not have to care about various tools e.g. both batch and streaming jobs can be performed in the same system. Apache Spark [235] and Apache Flink [70] are two prominent general-purpose Big Data Analytics systems. Apart from batch processing and streaming, they possess SQL-like expression languages (Spark: Spark SQL, Flink: Table), distributed graph processing (Spark: GraphX, Flink: Gelly) and ML libraries (Spark: MLlib, Flink: FlinkML). Both frameworks provide support for implementing iterative algorithms while keeping the data in memory and thus give faster convergence of iterative ML algorithms.

Spark has been widely used in this thesis and a separate overview of the system is given in 2.2.

## 2.2 Apache Spark

Apache Spark [235] is a parallel programming and execution framework for cluster computing that is fast and easy to use. In terms of speed, it’s much faster than well-known Google MapReduce [51] and its famous open-source implementation, Apache Hadoop [61]. One Reason for its agility is keeping the data in-memory with support for iterative processing. Even if the data doesn’t entirely fits into memory, it can spills out extra data to disk.

Before Spark, different distributed systems that included interactive queries, batch applications, iterative algorithms and streaming had to be managed using separate tools. For example, Hadoop had Hive for SQL processing, and there were Mahout for ML and Apache Tez for interactive and batch processing, but all of these came as separate tools and had to be unified by the user to make up a complete analysis pipeline. In contrast Spark comes with a complete Big Data Analytics stack, which includes Spark SQL for SQL processing, Spark Streaming for real-time processing, MLlib for ML, and GraphX for graph processing, all in a single
<table>
<thead>
<tr>
<th>Components/ Terminology</th>
<th>Spark Streaming</th>
<th>Flink Streaming</th>
<th>Storm</th>
<th>Samza</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Element</td>
<td>RDD transformation or action</td>
<td>DataStream transformation or action</td>
<td>Spout or Bolt</td>
<td>Task</td>
</tr>
<tr>
<td>Graph Edge</td>
<td>Implicitly defined by RDD transformation or action</td>
<td>Implicitly defined by DataStream transformation or action</td>
<td>Stream</td>
<td>Kafka Topic</td>
</tr>
<tr>
<td>Message Abstraction</td>
<td>RDD</td>
<td>DataStream</td>
<td>Tuple</td>
<td>Envelope</td>
</tr>
<tr>
<td>Primary Implementation language</td>
<td>Scala/Java</td>
<td>Scala/Java</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>DAG</td>
<td>Stream Processing Job</td>
<td>Stream Processing Job</td>
<td>Topology</td>
<td>Samza Job</td>
</tr>
<tr>
<td>Task Scheduler</td>
<td>Mesos, Yarn</td>
<td>Job Manager on top of Mesos/Yarn</td>
<td>Nimbus, Yarn</td>
<td>Yarn</td>
</tr>
<tr>
<td>Communication Scheme</td>
<td>Netty</td>
<td>Netty</td>
<td>Netty</td>
<td>Kafka</td>
</tr>
<tr>
<td>Message Delivery</td>
<td>Pull</td>
<td>Pull</td>
<td>Pull</td>
<td>Pull based Push</td>
</tr>
<tr>
<td>Fault Recovery</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Processing Guarantee</td>
<td>Exactly once</td>
<td>Exactly once</td>
<td>At least once</td>
<td>At least once</td>
</tr>
<tr>
<td>Recovery Method</td>
<td>Write ahead log</td>
<td>Check-pointing</td>
<td>Upstream backup</td>
<td>Check-pointing</td>
</tr>
<tr>
<td>Time Window</td>
<td>Batch based</td>
<td>Yes</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
unified package. Furthermore, Spark has a much larger API than Hadoop (which only had the map and reduce functions). Spark applications can be implemented on local machines and later run remotely on a larger cluster in a similar fashion. These facilities make Spark a more general-purpose framework for massive parallel processing.

To use Spark, the user writes a driver program that controls the overall flow of the program. The driver program contains some Spark constructs and parallel operations on these constructs. Resilient distributed datasets (RDDs) are the main programming construct of Spark. RDDs are a collection of datasets partitioned across the cluster that can be manipulated in parallel by applying different operations available in the Spark API. Two types of operations can be applied on RDDs, namely transformations and actions. Transformations are operations that, when applied to datasets, produce new datasets. As an example, RDD.filter filters the existing dataset on the basis of specified criteria and produces a new dataset. In contrast, when Actions are applied to datasets, they produce results that are returned to the driver program, e.g. RDD.count returns the number of elements in the RDD.

Spark’s other advantage over Hadoop is its resource-saving fault tolerance. RDDs remember all their previous transformations, thereby creating a lineage. If one of the partitions of an RDD is lost, it can restore itself by transformation backward through the lineage. The method saves on I/O in comparison to other fault tolerance approaches, such as the replication used by Hadoop.

As well as normal variables, Spark provides support for shared variables. In clustered environments, each node normally has a separate copy of each variable but, in some cases, shared variables are needed. There are two types of shared variables: broadcast variables and accumulators. Broadcast variables can be used when we need to share a large read-only file on all nodes, rather than sending a copy of it with the associated tasks. Accumulators are used when we need to implement shared counters for parallel sums.

2.2.1 Spark Streaming

Spark Streaming [236] is a component of the Spark Big Data Analytics stack for real-time applications. Traditional distributed stream processing systems provided fault tolerance either through replication or upstream backup. The replication approach required double the amount of hardware resources, and the upstream backup was slow because the whole system had to wait for a new machine in the case of a machine failure. Spark introduced the D-Stream programming model to overcome these problems.

D-Streams consider streaming computations into sequences of deterministic batch computations over small time intervals. The input for each interval is stored across the cluster. At the end of a particular time interval, parallel operations (like map, reduce and groupby) are performed on the input datasets to produce intermediate or output representations. RDDs are used to store these datasets on clusters.
2.3. CLOUD COMPUTING

With RDDs able to regenerate themselves from their lineage, the conventional fault tolerance (i.e., replication) is not required.

D-Streams also have the advantage of unifying streams with batch processing and provide a fault tolerance strategy similar to that of batch systems but at a low cost.

2.3 Cloud Computing

While considering cloud computing, we need to understand the characteristics of cloud computing, the services provided by clouds, the service provisioning techniques for clouds and the personnel involved in using these services.

2.3.1 Definition

Various definitions of cloud computing exist. NIST provides a comprehensive definition of cloud computing as follows.

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models."

2.3.1.1 Essential Characteristics

The essential characteristics of cloud computing are shortly described in Table 2.2

2.3.1.2 Service Models

Cloud services are divided into three major categories, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The cloud services pyramid is shown in Figure 2.2

- **IaaS** - IaaS is the most basic cloud service capability. It provides hardware infrastructures to the consumers e.g., virtual machines, storage, processing and network services. IaaS layers abstract the details of the hardware infrastructure from the consumers e.g., location, scaling and data partitioning. Some examples of IaaS are Cisco MetaCloud, Microsoft Azure, Amazon Elastic Cloud (EC2) and Google Compute Engine.

---

2 https://azure.microsoft.com/
3 https://aws.amazon.com/ec2/
4 https://cloud.google.com/compute/
CHAPTER 2. BACKGROUND

Table 2.2: Characteristics of Cloud Computing

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-demand self-service</td>
<td>Unilateral provisioning of compute power automatically as and when required without human intervention</td>
<td>network storage and server time</td>
</tr>
<tr>
<td>Broad network access</td>
<td>Network capabilities provide through standard techniques</td>
<td>access by mobile phones, tablets, laptops, and workstations</td>
</tr>
<tr>
<td>Resource pooling</td>
<td>Dynamic assignment and reassignment of physical and virtual resources without customer knowing the location of the resource</td>
<td>resources include storage, processing, memory, and network bandwidth</td>
</tr>
<tr>
<td>Rapid elasticity</td>
<td>automatic scalability of resources rapidly both inwards and outwards</td>
<td>provisioning or decommissioning of nodes, servers or instances</td>
</tr>
<tr>
<td>Measured service</td>
<td>automatically control and optimize resource use by leveraging a metering capability</td>
<td>measuring storage, processing, bandwidth, and active user accounts</td>
</tr>
</tbody>
</table>

- **PaaS** - PaaS provides users with the capability to deploy operating systems and applications on top of the cloud infrastructure in an isolated environment, called a container. The end user only controls the deployed applications, and maybe the configuration settings for the containers, but not the underlying hardware. Examples of PaaS are Aprenda[^5], Google App Engine[^6] and Amazon Beanstalk[^7].

- **SaaS** - SaaS refers to the capability to run consumer applications on the provisioned hardware. The applications can be accessed through either a thin client or a program interface. Examples of SaaS are Workday[^8], Salesforce Customer Relationship Management (CRM[^9]) and Google GSuite[^10].

[^5]: https://apprenda.com/
[^6]: https://cloud.google.com/appengine/
[^7]: https://aws.amazon.com/elasticbeanstalk/
[^8]: https://www.workday.com/
[^9]: https://www.salesforce.com/crm/
[^10]: https://gsuite.google.com/
2.3. CLOUD COMPUTING

2.3.1.3 Deployment Models

Cloud services are provided to the end users in the form of various cloud deployment models: private cloud, community cloud, public cloud and hybrid cloud.

- **Private Cloud** - The private cloud is available for exclusive use by a single organization and may be used by multiple divisions within that organization. The cloud infrastructure may be owned by the same organization, a third party, or a combination of both and may be located at or away from the premises of the cloud consumer.

- **Community Cloud** - The community cloud is available for exclusive use by the members of a community with similar goals/interests. The cloud may be owned and operated by one or more organizations in the community, a third party, or a combination of them and may be located at or away from the premises of the cloud consumer.

- **Public Cloud** - The public cloud is provided for use by general public. The cloud may be owned by businesses, academic or governmental organizations, or a combination of these, and located on the premises of the cloud provider.

- **Hybrid Cloud** - The hybrid cloud is a combination of two or more different cloud infrastructures i.e. private, public or community infrastructures. The hybrid cloud is a “fit for purpose” approach, i.e. it is designed to be the right solution for the current job and depends on the needs of the users. Hybrid cloud technology offers communication between the combined cloud infrastructures, as well as portability of applications and data between the independent cloud infrastructures that make up the service.
2.4 Microservices

Microservices is a style of software architecture inspired by service-oriented software framework that arranges an application as a set of loosely coupled services [84]. The core idea is to provide better modularity and independent execution of services. Although mainstream languages like Java, C++ and python provides some modularity through constructs like classes and methods, they create a single executable software known as a monolith. The modules in a monolithic architecture share the resources of a machine, hence they cannot execute independently.

Contrary, microservices are structured as a collection of multiple small cohesive services that execute independently. Each service is self-contained and implements a single functionality. The services communicate among them through stateless messages where each pair of request and response is an independent transaction. Microservices have many advantages. They are easy to develop and maintain in autonomous, cross-functional teams with increased productivity and speed.

2.4.1 Deploying Microservices

Previously, the microservices were hard to deploy, but the advent of containerized environments and container orchestration has made it easier and given rise to the microservices architecture.

2.4.1.1 Docker Containers

A Docker container [224] image is a standalone, lightweight, executable package of software that contains all the dependencies required to execute an application: code, runtime, system libraries, system tools and settings. At runtime, the Docker images transform into Docker containers and run on Docker Engine. Docker containers are available for both Windows and Linux machines and regardless of the type of infrastructure, Docker containers will run the same and provide the same functionality.

2.4.1.2 Kubernetes

Kubernetes is an open-source orchestration tool for deploying containerized applications [116]. In today’s world of internet, most software applications functions in a distributed way and need to be reliable and highly scalable. Kubernetes is the tool that provides functionality needed to build and deploy such reliable, scalable distributed applications. Kubernetes provides self-healing for reliability: a container is restarted on failure, containers are replaced and rescheduled if a node dies, a container is killed if it does not respond and then restarted. Kubernetes can easily scale applications up or down on need. Scaling can be done by either a manual request by a user or automatically based on the CPU usage.
2.5 Machine Learning and Big Data Analytics

Machine learning (ML) is a type of data analysis that makes computers learn and find hidden value in data without being explicitly programmed to do so – this is done by automating the process of analytical model building. Currently ML is playing a key role in providing insights and extracting value from large datasets. However, without access to large datasets, ML may not be of much use because the machine learning process requires a lot of training data. Thus using ML and big data together can benefit researchers and businesses by bringing forth the value hidden in the data.

2.5.1 Types of Learning

Dependent upon the type and level of training data and the method by which the training data becomes available, two ML types exist: supervised learning and unsupervised learning.

2.5.1.1 Supervised Learning

In supervised learning, labelled training data is available to the model. The labelled data includes the input object and the relevant output value, commonly known as the label. Once the model is trained using the labelled data, it makes predictions on all the unseen examples [113]. This is the most common approach used in classification, regression and ranking algorithms.

- **Classification** - In classification, the unseen examples are divided into two or more classes using a model trained on data that has already been categorized and labelled into two or more classes. An algorithm that performs classification is generally known as a classifier and refers to a function that learns a mapping from input $x$ to output $y$ [165]. An example of a classification problem is spam filtering, where the classifier aims to separate spam email from the rest of the email. Another example is tumour classification which seeks to accurately classify tumours as malignant(cancerous) or benign(non-cancerous).

- **Regression** - In regression, the regression function learns a mapping from inputs $x$ to a range of output variables $y$. The only difference, from classification, is that a continuous response variable $y$, rather than a fixed number of output classes, exists in the regression. Regression analysis helps to understand the dependence of the output variable $y$ on the input variable(s) $x$. Examples of regression analysis are predicting the sale price of a house in a particular region, and predicting future currency trends based on current market conditions and other relevant information.


2.5.1.2 Unsupervised Learning

In unsupervised learning, labelled training is not available to the model and instead the model has to learn a function to predict the hidden structure of the data. Unsupervised learning solves the main problem of density estimation in statistics [131]. Commonly solved problems in unsupervised learning are clustering and dimensionality reduction.

- **Clustering** - In clustering, the algorithm is provided with an unlabelled dataset and is supposed to automatically group the data into similar clusters. Two common goals in clustering problems are to find the existence of subpopulations in the data, and to estimate which cluster each data point belongs to. An example of clustering is grouping together consumers by their interests for e-commerce and sending them appropriate advertisements [58].

- **Dimensionality Reduction** - Processing high dimensional data is complex and expensive, therefore high dimensional data is generally converted to a lower dimensional subspace by decreasing the number of parameters that hold the character of the data. Two common approaches used for dimensionality reduction are feature selection and feature extraction [151]. In feature selection, a subset of features is selected from the original features, whereas in feature generation, a new set of features is generated based on the original feature set by reducing data from the higher dimensional space to a lower level subspace.

2.5.2 Machine Learning Algorithms

Machine learning is a vast field and involves a range of algorithms. Here we present some of the most popular ML algorithms for the problems described in Section 2.5.1.

- **Linear Regression** - Linear regression is a supervised ML algorithm [165] that identifies the relationship between the input variable $x$ and the real-valued output variable $y$. The form of the input variable is linear and the learned model is based on linear predictor functions. Hence the model is a linear model and the method is known as linear regression. The aim of the algorithm is to find a straight best-fitting line, known as a regression line, through the already known points and then to predict the value of the output variable based on this line.

- **Logistic Regression** - Unlike linear regression which is used for predicting continuous variables, logistic regression is commonly used for predicting discrete variables i.e. in classification problems [165]. For example, in a binary classification, we try to predict the probability that a given example belong to class “0” as against the probability that it belongs to class “1”.
2.5. MACHINE LEARNING AND BIG DATA ANALYTICS

- **K-Means**: K-means \[112\] is an unsupervised clustering algorithm. As a clustering algorithm, the aim is to form clusters based on similarities in the data. The K-means is an iterative algorithm and forms clusters using two steps during iterations by assigning each point to a cluster and then moving the cluster centroid. Consider a problem where we want to have two clusters. The algorithm selects two random points as cluster centroids. Each point is assigned to either cluster one or cluster two based on its distance from the cluster centroids. Then the centroid of each cluster is moved to the mean of the points that were assigned to that particular cluster. These two steps are performed iteratively until the assignment of each point does not change any more and until the centroids do not move any further, i.e. when the algorithm has fully converged.

- **Support Vector Machines**: SVMs are supervised ML algorithm developed by Cortes et. al. \[78\] for two-group classification problems, although they later became popular for solving other problems, like regression and novelty detection. An SVM includes a training algorithm that uses training examples, which are already marked as belonging to one class or the other, to build an SVM model that predicts which category the new examples should be assigned to. An SVM model represents training examples as points in space, dividing these examples into categories by a clear gap, which is as wide as possible. In addition to linear classification, SVM can also perform non-linear classification using the kernel trick originally proposed by Aizerman et. al. \[43\]. Most of the non-linear classification algorithm is the same as the linear SVM algorithms. The only difference is that a non-linear kernel function is used instead of every dot product, thus enabling the algorithm to fit the maximum-margin hyperplane in a transformed feature space.

- **Random Forests**: Random Forests are an ensemble ML method based on decision trees used for classification and regression problems. Decision trees tend to overfit and thus random forests are used to train multiple models, thereby removing overfitting. The algorithm computes and outputs the mode of all the classes in a classification. In the case of regression, it outputs the average of the individual predictions by each decision tree. Methods using random forests have shown good predictive accuracy \[71\] and have been successfully applied to many applications e.g. body pose recognition \[194\].

Some other widely used ML techniques are Matrix Factorization or SVD, Naïve Bayes and Artificial Neural Networks.

### 2.5.3 Conformal Prediction

Conformal Prediction is a framework that complements the prediction from an ML algorithm with a valid measure of confidence \[217\]. Conformal prediction can work in combination with almost any underlying regression or classification algorithm,
e.g., support vector machines, gradient boosting, neural networks, and random forests. In the case of classification models, conformal prediction produces a set of labels, e.g., in binary classification it produces four possible prediction sets \{0\}, \{1\}, \{0, 1\} and \Ø (the empty set). Although the output is a region or multi-classed, rather than a point prediction, the main benefit of the technique is the validity of the model with a user-provided confidence threshold. For example, in a binary classifier the true positives are not excluded more than the confidence threshold on average, e.g., if the confidence level is 90%, then in only 10% of the cases will the true positives be excluded.

2.5.3.1 Inductive Conformal Prediction

A common set-up for conformal prediction is the transductive conformal prediction approach (TCP). In TCP, the model is retrained for each new observation. However, this is computationally quite expensive, especially for problems with large datasets, and therefore an inductive or batch setting, called Inductive Conformal Prediction (ICP) [174], has become popular.

The way that ICP works in a classification setting is quite simple. Firstly, a training set and a test set of examples with labels are required. The training set is divided into a proper training set and a calibration set. The training set is employed to train a model using any underlying algorithm. The calibration set is then used to measure a nonconformity score for each observation in the calibration set, which is a measure of “how different the current example is compared to the training set”. The model then produces predictions for the examples in the test set, and, for each class label \(l = 1, \ldots, k\), a p-value of \(x\) for class \(l\) is computed. If the p-value for the class label \(l\) is greater than \(\varepsilon\), then that label is added to the prediction set. Using this approach, it is guaranteed that on average the true label of \(x\) will be present in the prediction set with probability \(1-\varepsilon\) [174].

2.5.4 Machine Learning Libraries for Big Data Analytics

Most of the batch processing systems and hybrid systems discussed in Section 2.1.2 include scalable ML libraries. An overview of these libraries is given in Table 2.3. Additional miscellaneous ML algorithms (such as feature extraction and optimization) are available in the libraries mentioned in the Table.

- Apache Mahout- Apache Mahout [4] is part of the Hadoop Ecosystem and initially only included ML algorithms for MapReduce. Later the algorithms were optimized for Spark, H\(_2\)O and Flink. The ML algorithms that are available in Mahout are collaborative filtering, classification, clustering, dimensionality reduction and some miscellaneous algorithms which are mostly deprecated. The algorithms available in Mahout scale well with the increasing size of data [188]. One commonly reported characteristic of Mahout ML algorithms is extensibility. [144][111] reports successful add-ons using Mahout algorithms as the baseline.
### Table 2.3: Overview of Machine Learning Libraries and Available Algorithms

<table>
<thead>
<tr>
<th>Programming Languages</th>
<th>Mahout</th>
<th>MLlib</th>
<th>FlinkML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>Java, Scala, Python</td>
<td>Java, Scala</td>
</tr>
<tr>
<td>Supported Processing Framework</td>
<td>MapReduce, Spark, H2O, Flink</td>
<td>Spark, H2O, Spark, H2O</td>
<td>Flink</td>
</tr>
<tr>
<td>Version</td>
<td>0.13.0</td>
<td>2.2.0</td>
<td>1.3.2</td>
</tr>
</tbody>
</table>

#### Classification and Regression Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mahout</th>
<th>MLlib</th>
<th>FlinkML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Regression</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Hidden Markov Models</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Random Forests</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Gradient Boosted Trees</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Ridge Regression</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Isotonic Regression</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>k-Nearest neighbors join</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

#### Clustering

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mahout</th>
<th>MLlib</th>
<th>FlinkML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canopy Clustering</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K-Means Clustering</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Fuzzy K-Means</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Streaming K-Means</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Latent Dirichlet allocation</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Bisecting k-means</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Gaussian Mixture Model</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Dimensionality Reduction

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mahout</th>
<th>MLlib</th>
<th>FlinkML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singular Value Decomposition</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Lanczos Algorithm</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Stochastic SVD</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Principal component analysis</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>QR Decomposition</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Collaborative Filtering

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mahout</th>
<th>MLlib</th>
<th>FlinkML</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Based Collaborative Filtering</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Item-Based Collaborative Filtering</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Matrix Factorization with ALS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Matrix Factorization with ALS on Implicit Feedback</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Weighted Matrix Factorization</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>
• **MLlib** - MLlib [160] is part of the Spark Big Data Analytics stack and includes many ML algorithms, such as classification, regression, clustering and collaborative filtering, along with useful tools such as featurization, ML pipelines, statistics and linear algebra utilities. MLlib is an open-source project and has a rapidly growing community of developers. It has been successfully used for various parallel ML projects, e.g., Capuccini et al. [68] presents an MLlib-based distributed conformal prediction implementation for valid confidence estimation for large dataset problems and shows the validity and scalability of the algorithms using two large datasets.

• **FlinkML** - FlinkML [11] is an effort from the Apache Flink developer community. The goal of the library is to support developers to build scalable ML systems with minimum effort by utilizing the ETL capabilities already extant in the Apache Flink Ecosystem, thus creating ML pipelines without the need for external tools. FlinkML contains basic ML algorithms and utilities e.g., distance matrix and cross validation.

The following more specific Scalable ML libraries also exist.

• **SAMOA** - Apache Samoa [164] is a distributed streaming ML framework. Unlike the previous three systems, it does not contain a generic distributed stream processing framework and only concentrates on the ML Streaming Algorithms. The goal is to have an uncoupled distributed streaming ML framework that can run on any of the distributed stream processing engines (like Spark Streaming, S4, Storm and Samza) without handling the complexity of the underlying stream processing frameworks. Samoa allows developers to implement the algorithms once and execute them on any distributed streaming framework.

• **TensorFlow** - TensorFlow [35] is an open-source ML library from Google which utilizes data flow graphs for modelling. The graph node performs mathematical operations whereas the edges are multidimensional arrays of data i.e. tensors that flow in between operations. With TensorFlow, it is rather easy to implement, train and deploy neural network models that scale up from mobile devices to multiple servers, which is especially useful for large dataset problems. The scaling and distribution is made possible by the ability of TensorFlow to perform partial subgraph computations i.e. the partitioning of neural networks.

### 2.6 Cheminformatics

Computer technology has been utilized by chemists for a long time, but it was only recently in the 1990s that the term cheminformatics or chemoinformatics emerged. The first formal definition of cheminformatics [66] was given by F.K.Brown in 1998 as follows:
“The use of information technology and management has become a critical part of the drug discovery process. Cheminformatics is the mixing of those information resources to transform data into information and information into knowledge for the intended purpose of making better decisions faster in the area of drug lead identification and optimization.”

A more general definition is given by Paris and cited by Warr [222]: “Cheminformatics is a generic term that encompasses the design, creation, organization, management, retrieval, analysis, dissemination, visualization and use of chemical information”.

2.6.1 Drug Discovery

The process of drug discovery to fulfill the requirements of unsolved health disorders is one of the main efforts being undertaken to better the quality of human life. While cheminformatics techniques have been applied in various fields (such as the paper, pulp and dye industries), the most prominent use has been in the pharmaceutical industry and medical science for the purpose of drug discovery [214].

Figure 2.3: The drug discovery process

Figure 2.3 illustrates the general approach to drug discovery and development [67]. The process begins with deciding which disease to treat and identifying an appropriate molecular target which, when modified in an appropriate way, will have beneficial effects on a patient suffering from that disease. The next step is to find other molecules that hit or bind to the target molecule. Many molecules will bind but have no effect, and hence will not be considered for use as drugs, so the possibilities need to be narrowed down to those molecules which are likely to have the required effect – they are known as leads. Many leads may have the right effect on the target molecule, but might not have all the properties necessary to make them suitable as medicines. For example, they might not be absorbed well by the body and therefore medicinal chemists would seek to modify those molecules to make them more drug-like. This process is called lead optimization. Once a molecule is identified that might make a good drug, the next step is to evaluate its effectiveness and toxicity using a range of in vitro and in vivo animal tests. The stage is known as preclinical testing. Once detailed information has been gathered to show that the drug is likely to be safe, it can then be administered to humans.

The drug discovery process is expensive and can take many years. Advancements in computing technology are one of the main reasons behind the development of cheminformatics. They have made it possible to perform high throughput in silico
drug discovery processes in parallel to make the process agile. More details are provided on this topic in [chapter 4](#).

### 2.6.2 Virtual Screening

High-throughput screening (HTS) is a well established approach for identifying lead molecules for drug discovery. It includes screening a large number of chemical compounds against a target using an automated bioassay [156]. An alternative approach is in silico (or virtual) screening, where large virtual chemical libraries are screened against a target receptor using computational methods [31, 201, 189] to find structures that are likely to bind to a target. Virtual screening aims at reducing the time and the cost involved in the drug discovery process.

#### 2.6.2.1 Ligand-Based Virtual Screening

When the 3D structure of the target is unknown, virtual screening relies on the physiochemical properties of the known active molecules that binds to the target of interest. One such method uses QSAR models as given in Figure 2.4.

![Figure 2.4: A two step process for a typical Ligand-based Virtual Screening using QSAR models: Step 1: QSAR model training based on the experimental activity of the molecules. Step 2: Screening through compound predictions using the QSAR model.](#)

The method is to develop QSAR classification or regression models using the structural properties of the compounds and their effect on the biological activity of the compounds [211]. LBVS with the QSAR models is a two step process i.e. Model Development and Screening. In the model development, the molecular structures are translated into molecular descriptors or signatures and are labeled as either an active or an inactive molecule based on the experimental activity. Next, a QSAR model is trained with the labeled molecules. In the screening, QSAR models are used to predict the activity of new compounds from the virtual library. Once the molecules are predicted, a post process step is performed e.g. to sort the molecules and find out the top most molecules.
2.6. CHEMINFORMATICS

2.6.2.2 Structure-Based Virtual Screening

Another method that is commonly used for virtual screening, when the 3D structure of the target is known, is molecular docking and scoring as shown in Figure 2.5.

Figure 2.5: A step by step process for a typical Structure-based Virtual Screening

The process starts by choosing a target. Then a set of ligands (candidate molecules for docking) and a receptor for the target are selected. Before docking, the ligands and the receptor need to be prepared. The preparation step includes the addition or the removal of certain chemical entities (that may cause hindrance in molecular docking). Once the ligands and the receptor are prepared, the ligands are docked against the receptor. In molecular docking, a docking algorithm is applied to find the best binding pose of the ligand (an ion or molecule that binds to a target receptor to form a coordination complex) in, e.g., the active site of a receptor, and a scoring function is used to evaluate the docking [136]. After docking, virtual screening may include post-processing steps e.g. a further filtering of the molecules using a certain criteria.

2.6.2.3 Off-Target Predictions

Drug-target bindings form the basis of drug discovery [232]. It is well known that a drug can bind to multiple target proteins [119] and may cause off-target effects [176, 180]. Drug discovery is an expensive process and late project failures are the major cause, contributing to the high costs [175]. Understanding the off-target effects of drugs can be very useful, especially in the early stage of drug discovery. To determine drug-target interactions, pharmaceutical companies and academic institutions involved in drug discovery apply different methods to detect drug-target interactions, including in vitro pharmacological profiling [64]. However an interesting alternative approach is to perform off-target in silico predictions using pre-built models. These predictions constitute into target profiles for ligands [72], which
helps in understanding off-target effects, as well as opening up new opportunities to predict the affinity of NCEs against a battery of targets.

2.7 Neuroinformatics

Neuroinformatics is the field of research that integrates neuroscience and computer science. According to Merriam-Webster Medical Dictionary [83], neuroscience is “a branch (as neurophysiology) of science that deals with the anatomy, physiology, biochemistry, or molecular biology of nerves and nervous tissue and especially their relation to behavior and learning”. Currently, neuroscience is a big area of research and now, in the data age, is producing huge amounts of data. The computer science part provides us with the opportunity to better understand the dynamics of the brain from the structure and functions that exist in the brain, by manipulating the large datasets that are being produced. International Neuroinformatics Coordinating Facility (INCF) [17] defines neuroinformatics as a field of research “that tackles the challenge of integrating information across all levels and scales of neuroscience. Tools and standards provide solutions for fitting together and analyzing data that comes from vastly different timescales, techniques and animals across all levels, from single cells to the whole brain. The field provides guidelines for recording information about data and how it is analyzed so that experiments can be reproduced and the data can be reused in new studies.”

Neuroinformatics leverages neuroscience by laying out standards and good practices for the informatics required in the area of neuroscience. The informatics support includes data acquisition, storage, analysis, visualization and modelling of neurological data.

2.7.1 Electroencephalography

Electroencephalography (EEG) is one of the main techniques used to measure the brain electrical activity [166]. EEG is mostly non-invasive as electrodes are attached to the scalp and not directly inserted into the brain. The electrical activity is recorded as multi-channel time series, which normally ranges from 128 to 2,000 samples per second per channel. The EEG device records the activity of many neurons simultaneously. The EEG output is not absolute but rather relative i.e. each EEG channel is normally computed by taking the difference between recordings of one or two reference electrodes [171]. This is particularly useful for recording and displaying very tiny electrical signals. In chapter 6, we focus on the analysis and modelling of neurological data in the form of a large EEG analysis.

2.8 Summary

This chapter has laid out a background to the topics which are needed to understand the rest of the chapters in this thesis. Big data and its characteristics were discussed,
and the main processing frameworks that are available were differentiated. We discussed Spark and its big data stack. ML was defined, and the best-known ML algorithms and libraries for Big Data Analytics were explained. The concepts of cheminformatics, drug discovery, virtual screening and neuroinformatics were then introduced. The next chapter will look at the related work undertaken by other researchers in the field of scalable analysis in the life sciences.
Chapter 3

Related Work

In this chapter, we present methods proposed earlier for the scalable analysis of large datasets in the life sciences for virtual screening, predicted target profiling and the analysis of EEGs.

3.1 Virtual Screening

A number of different methods have been proposed for virtual screening. We discuss these methods here.

3.1.1 Parallel and Distributed Methods

Virtual screening is trivially parallelizable on a per ligand basis, and many parallel approaches have been developed for it [207].

FlexX-PVM is a parallel version of a docking engine FlexX [179] used in a docking scoring functions comparison study [199] for handling large compound databases. FlexX-PVM is implemented in PVM (parallel virtual machine) [207] to run in parallel on heterogeneous environments and performs automatic scheduling, fault tolerance and defective ligand identification. It distributes single ligand-protein docking calculations on the available processors, configures hardware dynamically based on the ligand structure, starts/stops docking calculations by identifying defective ligand data and recovers failures of single processors. Automatic generation of the results file makes it possible to have the same results as the serial version.

High performance computing (HPC) and MPI have been utilized [239] [129] [240] for parallel virtual screening. DOVIS [239] is an application for large scale virtual screening on an HPC Linux clustered environment and uses AutoDock 3.0 as the docking engine. The researchers screened 500 to 1000 compounds per processor per day and overall successfully docked millions of molecules against a receptor. DOVIS also includes a GUI for job submission, visualization and querying the
CHAPTER 3. RELATED WORK

results. For parallel job submission in HPC environments, it includes a job queuing 
system. The job submission system automatically handles the three main steps 
in DOVIS i.e. the preprocessing stage where the system prepares receptors and 
molecules and transforms them into the relevant formats “pdbqs” and “pdbq” for 
AutoDock 3.0, the parallel docking stage, and the post-processing stage where the 
system collects the top hits from separate jobs into a single list.

DOVIS 2.0 [129] is an improved version of DOVIS with improved parallel screen-
ing, an efficient file system and improved usability. It also includes dynamic job 
control capabilities and dynamic load balancing without using a dedicated master 
CPU. The ligand set is divided into N blocks and distributed to N CPUs. Each 
CPU manages one ligand block at a time and thus enables the system to balance 
the load. Once a CPU finishes processing a ligand block, a new block gets allocated 
for processing. DOVIS 2.0 was updated with a newer docking engine i.e. AutoDock 
4.0 which comes with its own improvements [7]. The Autodock code was modified 
to provide support for a single energy grid file for several ligands, thereby saving on 
storage costs. By default, AutoDock docks only one molecule at a time and uses a 
separate energy grid file for each molecule. The output of the application is in the 
SD-File format, which has become the most recognizable format for molecule files.

An HPC-based task-parallel AutoDock implementation has been proposed in 
[89]. The first two steps i.e. the receptor ligand preparation and docking were 
parallelized using MPI. The post-docking steps, such as the filtering and sorting of 
molecules after collecting the results, were performed sequentially. For the post-
docking processing, the python glob() function was used – it provides fast access 
to datasets and thus parallel processing was not required. The experiment was 
performed on the Jaguar Cray XT5 Supercomputer at Oak Ridge National Lab-
atory using 65K processors. It took about 24 hours to complete the processing 
for docking the dataset of 1 million molecules. Reading millions of files from a 
directory is a very slow process as searching becomes expensive. A better strategy 
in this scenario is to store the files in hierarchical directories.

Another HPC-based virtual screening study [240] presents a master-slave data-
parallel pipeline, which includes rescoring top hits using a more accurate compute-
intensive method. First, the ligands and receptors were prepared. The ligands 
were docked into active sites of the receptor using VinaLC [121] and scores were 
computed. The docking was performed in a master-slave style where the mas-
ter is responsible for dividing the dataset and sending part of data to each slave. 
Each slave performs docking and sends scores back to the master for output. After 
computing the top hits, the top 20 hits were rescored using molecular mechan-
ics/generalized Born surface area (MM/GBSA) [120] to compute the free-energy of 
binding for the ligand-receptor complexes.

WISDOM is the first major research project for large-scale virtual screening 
on public grid infrastructures [126]. A CPU-intensive application was developed 
to create large data movement to test the grid operations and services. From a 
biological perspective, a large scale virtual screening was performed and 41 million 
compounds were docked to identify new inhibitors for a family of proteins produced
by the malaria parasite in humans. The project was deployed on the EGEE grid infrastructure for a period of 40 days on 1700 computers, which was equivalent to 80 years of CPU time. The key issue that was encountered was the stability of such a large scale grid infrastructure. Around 80% of the jobs submitted were successfully completed, and the jobs that were cancelled due to system instability were restarted manually. Using grid infrastructures offers the benefit of speeding up compute-intensive applications (like docking) by sharing resources. Other advantages are that database replication is possible, and that there are facilities for workflow management which make it possible to identify new biological targets. Data distribution and management features make the daily docking tasks easier. Also sharing computations and storage through grid infrastructures brings researchers from the developed and developing countries closer for better and speedier research collaboration.

[102] is another computational grid-based study for distributed molecular docking and virtual screening. Virtual screening was performed on Chemomentum [184], an open, collaborative grid computing environment for cheminformatics based on UNICORE grid middleware [92]. Chemomentum includes build-in facilities, like molecular docking, QSAR predictions and computational modelling. For the case study of grid-based virtual screening, the proteins HIV-1 and H5N1 influenza viruses were selected. AutoDock [7] and Glide [13] were used as docking engines, and the results were computed based on the high scores from both docking engines. For both receptors around 70K ligands were chosen by filtering out ZINC 7 [125], DrugBank [228] and NCI [20], and then the top hits for both receptors were reported. The study concluded that such collaborative grid environments are important for providing chemical tools, computational resources, scientific knowledge and expertise at a single location and predicting more properties of docked molecules using pre-trained QSAR models.

After the emergence of Google’s MapReduce in 2005 and its open-source implementation, Hadoop, around 2011, several studies [88, 152, 241] were performed using Hadoop, taking advantage of built-in fault tolerance and scalability. [88] is a basic study demonstrating the feasibility of the Hadoop framework by implementing a MapReduce-based AutoDock application for parallelizing virtual screening. The application was benchmarked on a private cluster by docking the DUD (Directory of Useful Decoys) dataset [124] against the human estrogen protein (PDB entry 1L2I [59]). The docking results were not affected by the parallel approach and produced the same results for the DUD dataset as those that were produced by other docking engines [122]. The utilization of the cluster by AutoDock was monitored using Ganglia [12]. Results suggested that the mapping tasks utilized 50% to 75% of the nodes that were provided.

Another Hadoop-based virtual screening study [152] proposed a modified hierarchical MapReduce strategy for small clusters. The strategy is especially useful in situations where a large cluster is not available due to expense, security etc. The hierarchical framework gathers resources from many small clusters and runs MapReduce jobs on these resources. The framework includes a central manager
that divides the dataset into blocks and load balances the data blocks on the small clusters based on the availability of resources on each cluster. Results from each MapReduce job on the small clusters are then returned to the global manager for final reduction. As a case study, instances of AutoDock were deployed on each of the clusters and the block set of ligands was docked using AutoDock. The load-balancing algorithm distributed the ligand dataset on multiple clusters and reduced the total execution time for the complete docking job.

Another approach [241] to using Hadoop MapReduce for large dataset problems is to use HDFS as a replicated distributed storage and then use MapReduce on top of that for data processing. The map function is used to dock molecules in parallel using many mappers on each node, and then the reduce function is used to collect the output of all the mappers from each node and produce the result.

Clustering is commonly used in cheminformatics to cluster similar compounds and predict the activity of compounds based on the known activity of other compounds in the same group. In [94], the authors implement a methodology for clustering-based molecular geometry classification that is both accurate and scalable. For scalability and load balancing, the clustering method was implemented using Hadoop MapReduce programming paradigm. Normally during protein-ligand docking, the ligands with the lowest energy are considered as the near-native conformers. In large numbers of ligand conformations that is not true because in some cases the protein-ligand energy modelling is inaccurate [143]. Therefore an alternative approach of hierarchical clustering is used and the results are clustered into already known reputed protein-ligand binding geometries. According to reported results [94], conformations for hierarchical clustering approach are better that the protein-ligand energy conformations, and they are as accurate as non-parallel clustering methods that are more time expensive. The method scaled linearly with the increasing numbers of ligand conformations.

Another strategy [155] used for compound identification in parallel is a MapReduce implementation of the Ward clustering algorithm. Two basic steps exist in the Ward clustering algorithm i.e. finding the closeness of the clusters and merging the two closest clusters into a single cluster. Each mapper instance searches the local minimum and sends it to the reducer instance. The reducer instance searches for a global minimum of all the local minimums and sends it to the relevant mapper to merge the two clusters. The benchmark study [155] reports a time saving of 17% for 3 mappers and 58% for 6 mappers.

After GMR and Hadoop, Spark has been the next parallel programming framework to emerge that utilizes MapReduce and has the ability to perform in-memory computing and provides novel fault tolerance. The existing studies showing Spark’s applicability and scalability for life sciences problems in general, and virtual screening in particular, are inadequate. Its performance was investigated with two virtual screening studies in Paper I and II [35, 153] showing its applicability and scalability for virtual screening in a clustered environment. The two studies are discussed in (chapter 4).
3.1. VIRTUAL SCREENING

3.1.2 Intelligent Iterative Methods

Iterative screening is one of the techniques used for virtual screening [145]. Rather than docking all the compounds in the dataset, a medium-sized dataset is docked and scored. On the basis of the docking scores in the first dataset, the next most active subset of the dataset is selected and docked. The purpose of iterative virtual screening is to reduce the number of docked molecules. ML can be useful [221] to create a model from the first docking and to predict the next dataset to be docked.

One such method is CerBeruS [90], which was developed for intelligent iterative screening based on hierarchical ward clustering, as discussed in Section 3.1.1. The dataset is divided into clusters based on the structural similarity of the compounds. An initial dataset is selected for the first iteration of screening from each cluster. The clusters are differentiated as being either active or non-active clusters. An active cluster must contain at least one active compound. Based on the first screening, a structure-activity relationship (SAR) model is created. Additional samples of active compounds are chosen by searching the database of as yet untested compounds based on the SAR model of the previous step. In the subsequent steps of the screening, all the compounds in the active clusters are retested. The process of screening and the searching database continues until a strong SAR had been reached. This strategy works by selecting compounds with a very strong similarity, and thus lacks diversity and may result in missing active compounds in the case where the initial dataset does not contain representatives from each active structural class [145].

An iterative virtual screening approach, based on a genetic algorithm, was presented in [223]. The general idea of the genetic algorithm is to iteratively improve the solution by having better characteristics in the next generation than those of the parents. The study identified an active thrombin inhibitor out of 160K compounds using the genetic algorithm iterative strategy. After virtual screening, the active compounds were screened in vitro against a thrombin receptor in each iteration that worked as a feedback loop function. They performed a total of 20 generations (iterations) and found a total of 400 reactions. The most active compound was found in the 18th generation. The top two active inhibitors that were identified had similar structural features to those of the already known thrombin inhibitors.

Two iterative virtual screening studies [146, 182] used a similar approach of hybrid in silico-in vitro screening. These studies take a medium-sized chemically diverse initial dataset and dock the dataset in silico using AutoDock. The top hits found from the in silico screening are further tested in vitro for inhibition. The verified inhibitors are then used for searching a larger database of 1 million compounds for similarities. The resulting compounds from the database search are then screened in silico with AutoDock and the top hits are tested in vitro again. The iterative process is repeated whenever new inhibitors are found. The two hybrid studies were able to find inhibitors with micromolar or better potency. Several other studies [237, 115, 114] have been conducted that use similar techniques of combining iterative screening with substructure searches and in vitro screening, and these have successfully yielded compounds with submicromolar potency.
Conformal prediction has been successfully used for moderate to small datasets in Quantitative Structure-Activity Relationship (QSAR) predictive modelling \cite{170,87}, complication risk prediction following a coronary drug eluting stent procedure (\(\sim 2K\) examples) \cite{55}, and anomaly detection of trajectories \cite{196}. Svensson et al. \cite{205} is another iterative virtual screening study based on conformal prediction. An initial dataset was selected for screening and docked. The results of the initial screening were used to classify the rest of the dataset using conformal prediction. The results from the strategy was checked against 41 receptors from DUD and DUD-E. The researchers reported that 57\% of the active compounds could be found by only docking 9.4\% of the compounds using the DUD ligand set of 2950 compounds using the conformal prediction approach.

We performed a similar study in Paper III \cite{40}, but on a much larger scale with 2.2 million molecules and used Spark as the parallel framework. Previously in Paper I and II \cite{153,38}, we had reduced the virtual screening time by parallelizing the docking step using Spark, but even then it was time consuming and therefore we further sped up the process by using ML models, namely SVM and conformal-prediction-based iterative models inferring the “low-scoring” ligands and neglecting them while docking only the “high-scoring” ligands \cite{40}. This work is discussed in Chapter 5.

### 3.1.3 Predicted Target Profiles

A common approach for building target profiles is to predict them using a number of QSAR models which are based on interaction values available for known active ligands in large interaction databases like ChEMBL \cite{103} and ExCAPE-DB\cite{203}.

Multiple studies and tools are available for building, predicting and employing target profiles. Yu et al. \cite{234} presented a systematic approach for predicting drug-target interactions from heterogeneous biological data using Random Forest and SVM. TargetNet \cite{231} is a web service for creating prediction-based drug-target interaction profiles using Naïve Bayes-based multi-target SAR models. In targetNet, the molecules can be predicted against 623 SAR models. Bender et al. \cite{57} employed a Bayesian-based method to prepare 70 QSAR models. The models were used to create the target profiles so as to predict adverse off-target effects of drugs. TargetHunter \cite{220} is another web-based tool for predicting target profiles using chemical similarities. The models were trained on ChEMBL data and successful predictions were made on examples taken from PubChem bioassays. The polypharmacology browser \cite{50} is another web-based tool for multi-fingerprint target prediction based on ChEMBL bioactivity data.

A key drawback with QSAR modelling studies is their reliance on experimental data from the large interaction databases. Such data tends to have a strong bias towards active compounds i.e. on-target or intended effects \cite{142}. Based on this, it is illogical to use ligand’s on-target binding data to build target profiles for understanding off-target effects. Therefore when studying the off-target effects of
drugs, it becomes useful to have another way than to simply examine data from the databases.

One, less common, method is to create models using molecular docking scores by virtual screening of ligands using a docking software and predict ligand-target binding affinity using the models. LaBute et al. [142] presented a method to predict side effects of drugs using scores produced by large-scale docking on HPC. Docking of 906 ligands was performed using AutoDock Vina and a set of 560 conformers was selected to train L1-regularized logistic regression models to predict 85 off-target effects. In a similar study, Wallach et al. [218] trained logistic regression model using docking scores from eHiTS [242] docking software for drug’s side effects predictions. Generating predicted target profiles using docking scores is uncommon as docking scores are not considered to represent real drug-target affinity, however large training datasets can cover this deficiency and allow to make better decisions.

We performed a similar study in Paper IV [37] for predicting target profiles where we built ML models based on in silico docking scores rather than on actual interactions given in interaction databases. The in silico approach is economically more effective, and in addition means that scores for large molecular libraries can be found in a comparatively shorter time, which makes it possible to build better ML models. A predicting target profile as a web service (PTPAAS) was implemented for target predictions using the web development framework Play 2.0 and was deployed as a microservice on OpenShift Origin. The work is discussed in the second half of Chapter 5.

3.2 Electroencephalography

EEG analysis plays an important role in diagnosing various neurological disorders, and in particular epileptic seizures [208]. Although varied techniques are available in the literature [210] for EEG analysis, the focus here is on scalable techniques performed using parallel programming frameworks and techniques for epileptic seizure detection in EEG data.

3.2.1 Parallel EEG Analysis

EEGs produces multiple time series for brain activity that can be processed in parallel. The basic approach is to use a compute cluster for parallel processing of EEG data.

A Beowulf cluster, a commodity computer-based cluster connected through a local network, was used in [230] for EEG processing in parallel. For processing EEG data on top of the cluster, a parallel and distributed compute engine was developed which acted as middleware between the processing applications and the physical layer of the cluster. The compute engine was responsible for task management and synchronization between various tasks. As a case study, parallel synchronization measurements among neural populations (that is, neuron ensembles in different areas of the brain) were implemented based on a method given in Li et al [147]. The
strategy consists of two steps, namely computing the correlations between all pairs of multivariate neural activity time series and creating a correlation matrix based on Random Matrix theory [227]. The synchronization measure of 0 means there is no synchronization, whereas 1 means perfect synchronization. The strategy is a compute-intensive task and as such was parallelized on a Beowulf cluster for early processing. Experiments were performed with an increasing number of nodes showing good scalability that became flat after reaching 8 nodes, indicating that parallel computing improves the execution efficiency where tasks are highly parallelizable.

In [133], A. K. Keshri et al. presents an MPI-based parallel algorithm for the detection of epileptic spikes. The data was divided into equal portions, given to processors and the same operations were performed on the data on each processor. A multiprocessor spike detection algorithm was implemented based on the Deterministic Finite Automata-based Spike Recognition (DFASR) [134] algorithm. The method was tested for EEG data representing periods of 60 minutes and the detection accuracy rate was 95.68%.

Graphical Processing Units (GPUs) have also been utilized for parallel EEG processing. In [53], EEG data was classified in parallel using spiking neural networks (SNNs). In SNNs [106], spikes are utilized instead of firing rates to achieve richer dynamics. Predefined set of images were shown as input data. EEG data was recorded in three categories: active, non-active and noise, and converted into images using Wavelet transform. The SNN was trained and tested using 100 images in parallel on GPUs. The task was divided into small parts to make it parallelizable. The parallel SNN on the GPUs was implemented in CUDA C. The six learning cycles of the algorithm took on average 1.92 seconds on Nvidia GeForce GT 520 GPUs. On more powerful GeForce GTX 580 GPUs, it took an average of only 0.23 seconds. The model classified EEG response images with an accuracy of 90.47%.

In [82], Deng et al. presented another method using GPUs to process non-stationary EEG data in parallel. For EEG processing, a parallel version of the Morlet continuous wavelet transform (MCWT) [108] was implemented. Three systems with different CPU and GPU configurations were set up for benchmarking. In all the three cases, the systems scaled well, but degradation was noticed when multiple GPGPUs were deployed on a single node. Another study [178] used GPUs to process EEG data using independent component analysis (ICA), a method to break multivariate signals into independent variables. The GPU-based optimization sped ICA up by a factor of 25. A recent study [206] presented the Neural Parallel Engine (NPE), a toolbox for processing neural signals in parallel on GPUs, which is provided as part of MATLAB. The toolbox includes many commonly used algorithms for processing neural signals e.g., spike detection and spike sorting. The authors claimed that NPE provides a 5 to 110 times improvement on CPU-based processing for various algorithms.

With the advent of cloud computing and Google MapReduce, several studies were performed for establishing the suitability of MapReduce for EEG analysis and processing. One approach is to analyse EEG data using ML in a cloud environment, e.g. Ericson et al. [91] classified EEG signals from multiple users by training a
3.2. ELECTROENCEPHALOGRAPHY

neural network with logistic regression, while MapReduce was used as the processing framework. A brain computer interface (BCI) application was implemented that enables physically disabled individuals to interact with the computer using EEG data in real time. A BCI application was developed in R (a free software for statistical computing), and R support was added to the runtime environment. The rest of the environment was based on Java and thus needed a lightweight Java-R interface to be implemented. The experiments showed that processing EEG data as data streams in real time was possible. The main bottleneck in terms of real-time EEG analysis was the classification step, while the training of the neural network model for logistic regression was relatively inexpensive.

In [60], Berrada et. al. presents a feasibility study for using Hadoop for the storage and processing of EEG data. As a case study, 46.5GB of EEG data was used, and the problem of feature selection was solved using KNN (k-nearest neighbours) as the feature classifier. The distributed KNN feature classifier was implemented using MapReduce. The authors concluded that Hadoop is suitable for both the distributed storage and processing of EEG data, and that it scales well with an increasing volume of EEG data. Cloudwave [127] is a similar project that presented a complete pipeline for processing neurological data in a cloud environment using the MapReduce framework. The scalability was tested on a 30 node Hadoop cluster and showed that the MapReduce-based pipeline scales well with an increasing number of nodes and volume of data. The benefits from using the Hadoop MapReduce approach were scalability, speed, reliable storage and fast access.

In the study [219], the authors implemented a parallel version of the Ensemble Empirical Mode Decomposition (EEMD) algorithm using MapReduce. They state that, although EEMD is an innovative technique for processing neural signals, it is highly compute-intensive and data-intensive. The results showed that parallel EEMD performed significantly better than the normal EEMD, and also verified the scalability of Hadoop MapReduce. Another study [89] used Hadoop and HBase-based distributed storage for handling large-scale multidimensional EEG data. The study was benchmarked on the Yahoo! Cloud Serving Benchmark (YCSB), and it checked the latency and throughput performance characteristics of Hadoop and HBase. The results suggest that these technologies are promising in terms of latency and throughput. However, at the time of the study, it was found that Hadoop and HBase were not mature enough in terms of stability.

3.2.2 Epileptic Seizure Detection

Machine learning algorithms, and especially support vector machines, have been widely used for seizure classification. [233] is a scalable patient-specific EEG seizure classification SoC(system-on-the-chip). The system generates patient-specific features and employs Linear SVM for real-time classification of seizures from non-seizure EEG data. The system can be scaled up to 8 channels. On average, the accuracy of seizure detection is 84.4%. In [56], Benatti et. al. presents a similar study that implements a scalable seizure detection algorithm on a multicore
architecture. The algorithm includes three main steps: dimensionality reduction, feature extraction and pattern recognition. Principal component analysis (PCA) was used for dimensionality reduction that converted the correlated data from $p$ sensors into linearly uncorrelated components. Feature extraction was performed using discrete wavelet transform (DWT) that provides information about the signal frequency content in the time domain. Pattern recognition was performed using SVM by classifying seizures from normal EEG with an accuracy of 98.9% and sensitivity (recall) of 0.85. Chisci et. al. [76] utilized autoregression models and SVM for predicting epileptic seizures from EEG data in real time. Feature extraction from online EEG data was accomplished via Auto regression models in combination with a least squares parameter estimator, and the classification into different neural states was performed using SVM. The Kalman filter (KF), an unknown variable estimator in a system, was applied to the SVM classification model, which enabled the model to achieve 100% correct seizure prediction and a very low false positive rate.

Qu et. al. [177] presents a patient-specific algorithm for seizure detection in long term EEG data. Seizure and non-seizure data was recorded for each patient and a modified nearest neighbour (NN) model was trained to do the classification. Features were extracted from the time and frequency domain. The features that were selected were the average wave amplitude, the average wave duration, the variation coefficient of the wave duration and the dominant frequency. Once the classifier was trained, it was used in later EEG sessions to check for seizures. An alarm was activated if the classifier detected a seizure. The NN algorithm had an accuracy of 100% with an average delay of 9.35 s after the onset of a seizure.

In [213], Tzallas et. al. presents a method for detecting epileptic seizures via time-frequency analysis. Time-frequency analysis was performed using the short-time Fourier transform and time-frequency distributions to calculate the power spectrum of each segment of the EEG data. The next step was feature generation using the power spectrum density (PSD) computed in the previous step. Time-frequency windows were created over the EEG data to compute the feature set. Principal component analysis was used to reduce the number of features, and three to four features were used in each experiment. Classification was performed using artificial neural network (ANNs) and was compared to other classification methods i.e. Naïve Bayes, KNN, decision trees and logistic regression. The ANN approach produced the best results, however it was also the most computationally expensive.

In addition to ML methods, mathematical transformational methods have been utilized for seizure detection and prediction in EEG data. [238] decomposes each EEG channel using wavelet packet transforms to find a separate seizure measure for each individual patient. The measure was based on probability density function of the EEG energy in seizure and non-seizure states and helped to select a frequency band where discrimination between seizure and non-seizure states was at a maximum. The combined seizure index (CSI), a normalized index, was developed using the measure for a selected frequency for each epoch of every EEG channel. The results had an accuracy of 90.5% and a median latency of 7 seconds. Another method
that has been used to successfully detect EEG seizures in real time is the short-time
Fourier transform (STFT). A comparison study [137] showed that STFT is better
than continuous wavelet transform (CWT), particularly for processing EEG data
in real time because of the method is lightweight and easy to process in real time.

Automatic real-time EEG Analysis can be beneficial in many scenarios. For
instance, manually monitoring continuous EEG data from epileptic patients to ob-
serve abnormalities is highly labour-intensive. However detecting epileptic seizures
in real time can provide timely alarms to the patient. It has been shown [138] that
automatic seizure detection in combination with an alarm signal can be useful to
alert a patient or a caretaker regarding the likelihood of a seizure. Methods for
EEG analysis in real time need to be lightweight as they have to process data as
soon as it is received, since they need to detect seizures in real time. Although
some studies exist that take care of the practicalities of real-time EEG seizure de-
tection, most of them strictly concentrate on only the accuracy with which seizures
are detected. Not many studies address the challenge of handling large volumes of
real-time EEG data e.g. if an EEG reading is recorded at a higher frequency and
with a higher number of channels, it can produce large datasets. In Chapter 6 we
present a method to handle large data challenges during the process of detecting
seizures from EEG data in real time.

3.3 Summary

In this chapter, we present previous work related to scalable analysis in the life sci-
ences. We present various methodologies previously employed for Virtual Screening
and EEG analysis. First we discuss the studies that utilize parallel and distributed
methods for Virtual Screening. Then we present studies that utilize iterative meth-
ods and various ML approaches for improving the efficiency of Virtual Screening.
Next we discuss different methods for developing predicted target profiles. Then
we look into studies that used ML algorithms and mathematical transformation
methods to solve the challenge of seizure detection and seizure classification. The
next three chapters will discuss the contributions of this thesis in detail by present-
ing novel methods to solve these challenges for large datasets in life sciences in a
scalable fashion.
Chapter 4

Parallel Modelling For Data Intensive Virtual Screening

This chapter is mainly based on Paper I [38] and II [153]. It presents an SVM based iterative parallel architecture for ligand-based virtual screening (LBVS) and a complete ready-made parallel pipeline for structure-based virtual screening (SBVS) using Spark.

4.1 Introduction

While today’s chemists and the pharmaceutical industry are privileged to have access to huge molecular libraries, like ZINC [125] and RCSB [23], these libraries cannot be utilized to their full potential due to their increasing size. The large molecular libraries, like any other large dataset, face the challenges of reliable storage, I/O management and reliable processing during hardware failures. With such challenges, the need for, and importance of, novel parallel methods and applications becomes more apparent.

Message Passing Interface (MPI) [77] has been the favoured parallel framework for a long period of time and many docking tools (e.g., OEDocking [21] and Multilevel Parallel Autodock 4.2 [167]) have successfully adopted MPI to parallelize virtual screening. However, MPI implementations have some limitations. MPI only provides an Application Programming Interface (API) to software developers, and it requires extra programming effort to handle other practical issues of parallel frameworks, such as fault-tolerance, load balancing and locality-aware scheduling. Consequently, many MPI-based applications depend on high speed network connections to offer scalability and, for the most part, the applications would not be able to complete their tasks if the hardware were to fail. As a result, organizations require access to high performance computing (HPC) facilities to run MPI applications effectively.

In 2005, Google produced its famous paper [81] on the MapReduce framework,
explaining the idea of using two basic functions, map and reduce, for processing large datasets. The significant benefits of MapReduce are that it offers the features of parallelization, fault tolerance, data distribution and load balancing united in a single framework, and that they are all transparent and already established for the user. In addition, using MapReduce opens up the possibility to use huge public cloud infrastructures for handling big data. Hadoop [61] is the most generally used open-source implementation of MapReduce. The Hadoop ecosystem includes HDFS, which provides scalable and fault-tolerant storage and can run on commodity hardware.

The MapReduce framework has its own drawbacks. It follows an acyclic data flow model and does not have built-in facilities for iterative programs where the same dataset needs to be accessed multiple times, thus neglecting an important class of applications that are based on iterative processing (such as graph processing, web ranking and machine learning [235]). Indeed, the absence of useful additives, such as in-memory computing, broadcast variables, accumulators and support for native workflows, makes it hard to develop scientific applications. Spark [235] is an open-source commodity computing parallel programming framework that covers the drawbacks of MapReduce, while keeping the features of fault tolerance and scalability.

Although Spark has been widely adopted in the business community, its significance in scientific applications is not so well understood. This thesis investigates its suitability and scalability for life sciences problems by implementing parallel ligand-based virtual screening (LBVS) with SVM in Paper I [38]. Once Spark’s suitability for virtual screening was established, Spark was used for implementing parallel structure-based virtual screening (SBVS) and an API for creating custom SBVS pipelines was developed, and is discussed further in Paper II [153]. With the LBVS and the SBVS studies, this chapter outlines the contributions of this thesis to the research question Q1 given in Section 1.2, i.e. a next-generation Big Data Analytics framework, Spark, was adopted for a life sciences problem and provided a ready-made solution for a large-scale problem with better resource utilization and speed-up.

The rest of this chapter is structured as follows. Section 4.2 presents LBVS method using SVM on commodity clusters. Section 4.3 covers SBVS methods for creating custom virtual screening pipelines and Section 4.4 summarises this chapter.

4.2 Using Iterative MapReduce for Parallel Virtual Screening: Feasibility Study for Spark

SVM [78] is a famous ML algorithm that was introduced for two-class classification problems and it has been a preferred algorithm for classification, regression and novelty detection problems for years. SVM has given better results for many applications in comparison to other techniques, but nonetheless it is a compute-intensive and resource-demanding technique and its complexity can be at least up to $O(n^2)$
Studies have been performed previously to speed up SVM-based applications. One method is to divide the data into small datasets and apply SVM to each smaller dataset in parallel.

In this work, a parallel SVM-based virtual screening model using Spark was implemented so that it runs on commodity clusters. In this implementation, the data and relevant libraries were distributed on the nodes of a cluster, performing virtual screening separately and in parallel on each dataset. LibSVM was used for the classification of molecular structures, while the application was implemented using Java and the Chemistry Development Kit (CDK). The method is available as an open-source repository on GitHub named spark4vs.

### 4.2.1 Virtual Screening and LibSVM

Virtual screening (VS) aims to filter out large collections of chemical structures in order to seek out potential candidates that are likely to bind to a drug target. The signature descriptor was used to convert chemical compounds into numerical form and the SVM models were used to rank the candidate structures according to the required predicted chemical properties including the ability to inhibit cancer cell lines, as well as solubility (logS) and partition coefficient (clogP), and also safety models such as AhR inhibition and HeRG inhibition as explained.

LibSVM is the most widely used and efficient SVM library. It includes features for performing classifications, regressions and distribution estimations. Here, LibSVM was used for pre-built classification and regression models applied to 21 million molecules available in the ZINC database in SDF format.

### 4.2.2 Parallel Ligand-Based Virtual Screening Implementation

Here the implementation of ligand-based virtual screening (LBVS) is discussed. As mentioned earlier, the molecules were available in SDF format and were predicted by LibSVM models. As a basic Spark investigation for virtual screening, only two SVM models were used to predict molecules.

#### 4.2.2.1 Tools and Infrastructure

Figure 4.1 presents the tools and infrastructure used in this work. An OpenNebula-based private cloud infrastructure was used as the physical layer. OpenNebula is an open-source piece of software for building and managing cloud resources that makes it possible to launch, delete, and clone VMs and images. The user has the options of using the web interface with a GUI or a command line interface (CLI) to manage the cloud environment. The GUI was found to be unstable during experiments, and hence the CLI was used for cloud management.
In addition to the private cluster, HDFS was used for data storage. Other than being fault tolerant when performing data replication, HDFS offers quick access to application data and is therefore most suitable for very large dataset problems. On top of HDFS, we used Spark for data processing and analysis.
4.2. USING ITERATIVE MAPREDUCE FOR PARALLEL VIRTUAL SCREENING: FEASIBILITY STUDY FOR SPARK

4.2.2.2 Architecture of Parallel Ligand-Based Virtual Screening

The LBVS approach was based on master-worker architecture as shown in Figure 4.2. The master machine acted as the NameNode for HDFS and the worker nodes were used as DataNodes for HDFS. Once the SDF files were copied to the private cluster, NameNode divided each SDF file into blocks; the data blocks were distributed and kept at the DataNodes, while the NameNode contained the metadata for the dataset stored at each DataNode. NameNode also had a file system namespace and controlled how the worker nodes can access the files. The number of blocks depends upon the size of the dataset. The number of blocks were calculated as the size of the dataset divided by the HDFS block size, which is normally 64MB in older versions and 128MB in newer versions of Hadoop. The data blocks contained the records, which are the smallest units that can be read by the worker nodes. In this case the structure of a single molecule acts as a record.

Once the dataset was distributed on the cluster, it was processed using Spark MapReduce with Spark master daemons and worker daemons running at the master and worker nodes respectively as shown in Figure 4.2. A single fat jar file was created using Apache Maven to transfer the application code and all the dependencies, including LibSVM and CDK.

The jar file was then provided to Spark through SparkContext (which is explained in detail in the next section), which deployed it to all the worker nodes. All the dependencies were local to the worker nodes and do not have to be remotely fetched from network, thus reducing the overall processing time. The worker nodes processed these datasets using MapReduce. Each worker node had mappers and reducers. Since each block was assigned to a mapper, the parallelism and number of mappers depends on the number of input blocks, whereas the number of reducers can be changed through the driver program. Once the worker nodes complete the processing, the results were sent back to the master node.

4.2.2.3 Workflow of the Driver Program

The pseudo code for the Spark-based LBVS is given in Figure 4.3. Before executing the driver program, some pre-processing steps were performed. Master and workers nodes were specified for Spark and HDFS in their specific configuration files by providing corresponding IP addresses. Many other Spark and HDFS configuration variables also need to be set in their respective configuration files. For example, the SPARK_WORKER_MEMORY variable can be used to allocate memory to each worker in the Spark cluster. Using a distributed file system, which was HDFS in this case, the data was then distributed over the worker nodes before processing.

In addition, a RunPrediction class was created which acted as the driver program. The overall parallel processing of the datasets was controlled through this class. In the initialization phase, predictive models were read and initialized into memory using the SignLibSvmModel class, which was created separately.
During initialization, various Spark system properties were also set. For example, by default Spark uses all the cores available on the machine, so if the user wants Spark to only use 4 cores, it can be achieved with the following code.

```java
System.setProperty("spark.cores.max", "4");
```

Spark accesses a cluster through a SparkContext. In the SparkContext, the master node IP and the jar file path that need to be passed on to worker nodes were provided. Next the SDF files were read into RDD using SparkContext. Spark can read any text files or files supported by the Hadoop ecosystem. SDF files cannot be read by the built-in `.textFile()` method, which supports records based on a single line. For reading multiligne molecular structures, it was necessary to have a customised input format which was achieved by extending the Hadoop FileInputFormat class and implementing a record reader. In this case, the SDFInputFormat class was used to read the SDF files. In the SDF file, each molecule was separated by `$$$$` and a new line. The following code shows how the SDF files were read, where the `endTag` represents the end of a molecule.

```java
byte[] endTag = "$$$$\n".getBytes( );
boolean status = readUntilMatch(endTag,true);
```

Once the RDD containing the SDF files was created, the map operation was applied to the RDD. By applying the map operation, the prediction function was
applied to all elements in the RDD. Internally the prediction function was executed separately in parallel on all the mappers on the worker nodes as shown in Figure 4.4.

In the map operation, CDK was used to read the molecules into IAtomContainer objects by parsing the string representations of the SDF records. Once parsed, the molecules were manipulated as normal using CDK. The signatures of the molecules were calculated and compared against the SVM models to find the predicted molecules. The output of the mappers was a (key, value) pair of molecules and an associated value of 1 or 0, where 1 indicates that the molecule is predicted to be active and 0 otherwise.

\[
\text{if (hergRes!=1) return new Tuple2(mol,1); else return new Tuple2(mol,0);}\]

A filter operation was applied on the RDD to select all the molecules with value 1. After filtering, a reduce operation was applied to get the aggregate of the successful molecules by reducing all the “1”s.

\[
\text{Integer total = winners.reduce(}
\text{new Function2<Integer, Integer, Integer>() }
\text{@Override}
\text{public Integer call(Integer a, Integer b) }
\text{retur a + b;}
\text{)}\]

Figure 4.4: Molecule Prediction with MapReduce
CHAPTER 4. PARALLEL MODELLING FOR DATA INTENSIVE VIRTUAL SCREENING

4.2.3 Results and Discussion

Experiments were undertaken to evaluate the performance of this LBVS implementation. In the LBVS experiment environment, virtual machines were launched and used on our OpenNebula based cluster. The virtual machines were based on AMD 64bit architecture. Each VM acted as either a master node or a worker node. The worker nodes had an internal memory of 2GB and 2 cores each, whereas the master node had an internal memory of 10GB and 2 cores for processing. Normally the master node does not need much memory but, when the number of worker nodes increases, it needs sufficient memory to keep track of all the worker nodes. Each node was installed with Debian OS.

4.2.3.1 Horizontal Scaling and Parallelization

This experiment evaluated the performance of virtual screening with an increasing number of nodes/parallelism. The dataset used for this experiment consisted of (approximately) 10GB of SDF files. All the files were equal in size, each having 564MB of data. The details of the experiment are illustrated in Figure 4.5.

![Increasing Number of Nodes and Computational Time](image)

Figure 4.5: Horizontal Scaling and Parallelization

This experiment started with 10 worker nodes, and increased by 4 nodes for each test. Initially, with the increasing number of nodes, the time decreased almost linearly: with 10 nodes, the virtual screening was completed in 49 minutes and with 26 nodes, it was completed in 25 minutes. This suggests that virtual screening can perform better with an increasing number of nodes. After 26 nodes, the time did not decrease as rapidly, and once the number of nodes was increased to 38,
the time levelled out to 21 minutes. This shows that parallelization can be done to certain limits. The reasons that the time did not decrease further with an increasing number of nodes are the time required for disk I/O and for network I/O – these increase with the increasing number of nodes and thus negate the effect of parallelization after 38 nodes.

4.2.3.2 Input Split Size

This experiment evaluated the performance of LBVS while changing the input split size. The number of nodes used for this experiment was 22 i.e. 44 cores. During the evaluations, the size of the dataset was increased for each test, with a total number of 5 different sizes being checked. The implementation was tested with two different input split sizes, 64MB and 128MB, as shown in Figure 4.6.

![Figure 4.6: Changing Input Split size](Image)

This test showed that increasing the input split size slows down the execution time of the implementation. The increased input split size decreases the level of parallelism since the number of mappers is inversely proportional to the input split size. With a 128MB split size, there were fewer mappers than with a 64MB split size and thus less parallelism. The experiment also answers the question of why the scaling was limited to a certain limit in the earlier experiment where the input split size was 64MB. It indicates that the previous experiment could have been parallelized further by decreasing the input split size to 32MB. The results also show that if the data size is bigger, one needs to have a bigger input split size and vice versa to optimize the execution time.
CHAPTER 4. PARALLEL MODELLING FOR DATA INTENSIVE VIRTUAL SCREENING

In this work, LBVS was implemented using Spark and its usability was established with an SVM-based molecular filtering case study. After Spark showed good scalability and usability for LBVS, it was utilized for developing an API for creating a complete pipeline for SBVS. The next section discusses this API and the implementation of SVBS using the API, as well as the deployment on public cloud resources.

4.3 Large-scale Structure-Based Parallel Virtual Screening on public cloud resources with Apache Spark

In drug discovery, a well-established approach for lead identification is high-throughput screening (HTS), which includes screening a large number of chemical compounds against a target using an automated bioassay \[101\]. HTS is an expensive process and only produces a limited number of hits. Also, a high number of false positives or false negatives can exist in HTS results. An alternative approach is structure-based virtual screening (SBVS). Compared to HTS, SBVS is a cheaper and faster method, in which virtual chemical libraries are screened against a target receptor using computational methods \[123, 75, 187\]. SBVS can be used to filter out molecules in the initial phase of drug discovery, making the discovery process cheaper.

A typical SBVS docking process starts with the preparation of a receptor. After the receptor preparation, the ligands available in the molecular library are docked against the receptor using the docking software of choice. The docking software produces a pose for each ligand, the best orientation of each ligand at the receptor’s binding site pocket, and a corresponding score. The ligands with the best scores are considered to have higher affinity in reality so they can be considered in later steps of the virtual screening process. The goal of virtual screening is to select the best hits from the molecular library and therefore the poses are sorted on the basis of their scores.

This section presents a method presented in Paper II for large-scale SBVS on public cloud infrastructures based on the MR framework as shown in Figure 4.7. This is an improvement on earlier work \[88\] where the researchers suggested that...
their AutoDockCloud method needed further enhancement to run on public cloud resources. To the best of my knowledge, this is the first reported work where SBVS has been successfully scaled over millions of molecules on public cloud infrastructures or on commodity clusters. The method is available as an open-source repository on GitHub named Spark-VS [28]. Ready-to-use examples of SBVS pipelines are also available in the repository.

**Figure 4.8: A typical SBVS Pipeline**

### 4.3.1 SBVS Pipelines

Using Spark-VS, the end user can build custom pipelines for virtual screening. Spark-VS provides a high-level API developed in the Scala Programming Language. Although the end user needs to have some knowledge of Scala and Spark, overall it is fairly easy to build a custom pipeline with Spark-VS. Figure 4.8 shows the flow of a typical pipeline for SBVS and Figure 4.9 presents code for a typical SBVS pipeline. In this work, a new SBVS Pipeline was created first, then the ligands file was read and the ligands were docked against the receptor file. The docking method takes three parameters: the receptor file in the OEDocking TK [21] binary format, the docking method and the search resolution method for the OEDocking software. All the docked poses were saved after the docking using .saveAsTextFile primitive. Docking is an expensive process and thus it is useful to save poses after docking. The .getTopPoses method was used to extract the top 10 hits based on the docking scores. Internally the method sorted the poses by scores to obtain the top hits. Once a pipeline is defined, it can be packaged like any other Spark application and is ready to be submitted to the Spark cluster. The effort required to deploy a Spark cluster is essentially a matter of running a set-up script.

#### 4.3.1.1 Data Input

By default, Spark can read text files or files supported by Hadoop framework. When Spark reads file into an RDD, it splits the file line by line. In contrast, the SDF files contain multilane 3D formats for each molecule. SDF files are useful as they make
it possible to store metadata for molecules e.g., molecule IDs, signatures or scores along with the 3D molecular structures [80]. A record reader was defined in Spark-VS, so it would be possible to properly read and split 3D molecular structures to process them across the cluster. Each instance of the docking program takes some time to initialize, therefore a parameter chunk size was defined that enables to set multiple molecules loaded into each RDD record. The chunk size has a default value set to 30; a chunk size that is too small will initialize many docking instances and a too large chunk size makes it harder to balance loads.

4.3.1.2 Parallel Screening

OEDocking TK [21] was employed as the docking software since it comes with an academic license. Using OEDocking TK C++ API, a light-weight docking executable was implemented that takes a chunk of molecules from standard input, docks the molecules to a specific receptor (i.e. adds them to each worker node using Spark addFile method) and returns the poses along with their scores to standard output. In Spark-VS, a map RDD operation was defined to pipe each RDD record to the C++ docking executable and get the results back via standard output. This method differs from [241, 88] where molecules were passed to, and received from, docking program through Hadoop local disk files. In contrast, this method was completely processed in memory, which has performance benefits over disk I/O.

Spark RDD’s API includes useful methods that were used for the post-processing step of SBVS, which comes after the docking step. For example, Spark’s .saveAsTextFile was employed to store poses in HDFS. It was necessary to sort the poses to get the top hits in terms of their scores. The Spark built-in .sortBy is an expensive operation for large datasets in a distributed manner due to the involvement of large network data shuffling. Rather than performing a distributed sort, an efficient workaround method was defined in Spark-VS, namely collecting (ID, score)-tuples for each pose in the driver machine and sorting them serially based on score. Once these pose tuples were sorted, the RDD filter operation was used to obtain the top scoring molecules by ID.
4.3. LARGE-SCALE STRUCTURE-BASED PARALLEL VIRTUAL SCREENING ON PUBLIC CLOUD RESOURCES WITH APACHE SPARK

To check the correctness of SBVS, 1000 molecules were randomly selected from the benchmark dataset and docked serially using OEDocking TK— the same scores were found during this process as those recorded during the parallel screening. The validation procedure is provided along with Spark-VS and is easily reproducible, running Spark in local mode.

4.3.2 Experiments

4.3.2.1 Environment Settings
Experiments were performed on a Spark Standalone cluster along with HDFS on infrastructure from the City Cloud [10], which is a public cloud provider. SparkNow [27] was used for hardware orchestration i.e. for creating images and provisioning virtual machines. A total of 21 nodes were deployed, each of them had 4 virtual CPUs (vCPUs), 8GB of internal memory, 20GB of ephemeral storage and 40GB of block storage. One node was used as the master and 20 nodes were used as worker nodes with a total of 80 CPUs in the cluster employed for actual processing. The parallelism was set to 80, so each CPU was given a single task. The environment was fully virtualized and mirrored a commodity compute cluster with respect to resources.

4.3.2.2 Benchmarking
The SureChEMBL molecule library [173] was used for the benchmarks, downloaded from ZINC [125] in ready-to-dock SDF format and docked against the HIV-1 receptor. The library contains ∼2.2 M molecules. HDFS was used to make the dataset available to the worker nodes. The HDFS block size was set at 64MB with a replication factor of 3. The dataset size on disk was relatively small, ∼8GB on disk, in comparison to the MR processing capability, however molecular docking is compute-intensive, unlike traditional MR applications, which justifies the applicability of Spark for molecular docking. The target receptor HIV-1 protease was available in PDB format [52] and was prepared using OEDocking TK in the OEB format.

4.3.2.3 Scaling and Speed-Up: Performance Evaluation
The first experiment investigated the scaling efficiency of Spark-VS, finding out the utilization of the processing units during molecular docking. The scaling efficiency is an important parameter for testing the performance of parallel frameworks, especially while using cloud resources as they are usually pay-per-use. For estimating the resource utilization, Spark-VS was run repeatedly over 25%, 50%, 75%, and 100% of the dataset, with limiting vCPUs used to 20, 40, 60 and 80 respectively. The Weak Scaling Efficiency (WSE) was then computed, i.e. the time taken by one processing element (20 cores in this case) to process one unit (25% or 1/4) of the dataset divided by the time taken by N processing elements to process N units
CHAPTER 4. PARALLEL MODELLING FOR DATA INTENSIVE VIRTUAL SCREENING

Figure 4.10: Weak Scaling Efficiency

(\text{where N = 1, 2, 3, 4 in this case}). The WSE for each run is shown in Figure 4.10. The trend curve in the figure was computed using 2nd degree polynomial interpolation. It is important to note that this WSE experiment demonstrated a scaling efficiency of 87%.

Another important factor for checking the performance of a parallel implementation is the speed-up, which is a comparison of the parallel running time against the running time on a single core. The speed-up is defined as $T_1/T_N$ where $T_1$ is the single core running time and $T_N$ is the parallel running time and $N$ is the level of parallelism. Simply put, the speed-up indicates how much faster a computation gets completed at a particular parallelism level. For computing $T_1$, first the running time for processing each molecule on the given vCPU was recorded. Figure 4.11 shows the histogram with equally spaced bins by running time for each of the 2.2M molecules. The serial running time, $T_1$, for each of the molecules, adds up to $\sim$634.7 hrs whereas the total time taken for molecular docking on 80 vCPUs was 8.9 hrs. Therefore, on 80 cores, a speed-up of $\sim$71 was achieved.

4.3.3 Discussion

These experiments suggest that the SBVS methodology scales well. A scaling efficiency of 87% and a speed-up $\sim$71 were obtained. In actual fact, both of these
results are closely related to each other. It is only possible to have good speed-up if the available resources are utilized efficiently. In Figure 4.10, it is notable that the WSE level almost becomes flat while increasing the number of vCPUs from 60 to 80. Also, the long running time keeps the analysis running overnight. For these two reasons, the extra costs required for bigger runs are not justified.

In Figure 4.11 it can be seen that the running time for each molecule varies a lot, and most of the molecules have a running time between 0.75 and 1.50. This observation is worthy of note because Spark did not have any information about the molecules and they were randomly distributed by Spark across the cluster. Knowing the cause of variability in the running time of each molecule and distributing the molecules accordingly could further improve the efficiency of parallel virtual screening, though it was not part of the current work and has been noted for future consideration and exploration.

A speed-up of 450 was reported by Ellingson and Baudry [88] for AutoDock-Cloud on a bare-metal based Hadoop environment. A bare-metal implementation is not comparable to SBVS where a completely virtualized environment was utilized. Most of the biggest public cloud providers, such as Google Cloud Platform [14]...
and Amazon Web Services [3], leverage their services on virtualized environments, hence it appears that MR applications should always be tested on virtualized environments. It was possible to compute the WSE for from the numbers given in the study. The average unit work time was 470 s, whereas the running time for parallel AutoDockCloud was 550 s. Hence, the WSE was computed as being equal to 85%, whereas for SBVS a slightly better WSE of 87% was achieved.

4.4 Summary

This chapter presented novel parallel methods for Virtual Screening on public cloud infrastructures using Spark. Research question Q1 (see section 1.2) asked whether next-generation Big Data Analytics frameworks can be adopted for life sciences problems and whether these frameworks be utilized for ready-made solutions to large-scale problems with better resource utilization and speed-up?

To investigate the first part of this question, a basic LBVS study was presented where two SVM models were used to filter out the predicted molecules. The suitability of Spark for parallel virtual screening was demonstrated through satisfactory horizontal scaling and hence the applicability of next-generation big analytics tools for life sciences problems was verified. To answer the second part of the research question, a parallel SBVS method, which enables end users to build custom virtual screening pipelines in cloud resources or commodity clusters, was presented. The performance experiments for the SBVS method displayed good scalability and speed-up and thereby opened up opportunities for virtual screening on large public cloud infrastructures, which is vital for organizations that cannot afford access to HPC facilities. Thus it was demonstrated that next-generation big data tools are definitely useful for creating ready-made solutions in the field of life sciences.
Chapter 5

Efficient Intelligent Modelling For Virtual Screening and Applications

This chapter is mainly based on Paper III [40] and Paper IV [37]. First, we present an efficient iterative virtual screening strategy that uses Spark and Conformal Prediction (CPVS) and then we present an approach for predicting target profiles for NCEs using docking scores as a web service (PTPAAS).

5.1 Introduction

In Chapter we demonstrated the importance of parallel virtual screening strategies. Although such parallel strategies allow large scalable implementations of virtual screening and ligand docking to work at a faster rate, all the ligands available in the libraries still need to be docked as we have no information available which indicates that particular ligands should be chosen and docked. As discussed earlier, the aim of virtual screening is to find lead candidates that show high affinity against a particular target. In fact, only a very small number of the high affinity ligands are present in large chemical libraries for any particular target and consequently a lot of time and computation power is consumed docking ligands of little importance. The docking time can be substantially reduced if “high-scoring” ligands can be identified with confidence in advance so the ligands expected to be “low-scoring” can be exempted from the docking computations. In order to achieve this, conformal prediction can be used to build models from the already-docked ligands and those models can be used to filter out the ligands likely to be “low-scoring”.

Such ready-made models are also useful for other end-user applications, e.g. to predict new compounds and indicate how they are likely to perform against a particular target, or to create complete target profiles for compounds through predictions for a list of targets. Again, it generally increases the efficiency to dock only the molecules that are likely to be of interest – these can usually be identified after checking the target profile and the affinity of particular molecules against
CHAPTER 5. EFFICIENT INTELLIGENT MODELLING FOR VIRTUAL SCREENING AND APPLICATIONS

several targets.

Here we first present a novel iterative strategy, CPVS, for distributed, structure-based virtual screening using Spark’s MLlib library, distributed conformal prediction [68] and support vector machines (SVM) [78]. Once CPVS was successful, we used it to build more ML models for the PTPAAS project, which is a web service for predicting target profiles of chemical compounds for a list of targets. The target profiles can be used to predict off-target effects, for example, in the early stages of drug discovery projects. The CPVS strategy (Paper III) and the PTPAAS (Paper IV) studies discussed in this chapter demonstrate our contribution to the research question Q2 listed in Section 1.2, i.e. we increased the efficiency of a Big Data Analytics method in the life sciences using machine learning and provided an intelligent end-user tool based on it.

The rest of the chapter is structured as follows. In Section 5.2, we discuss Iterative virtual screening with conformal prediction (CPVS). In Section 5.3, we discuss a web service for predicting target profiles (PTPAAS) for NCEs and in Section 5.4, we summarise the chapter.

5.2 Efficient Iterative Virtual Screening with Apache Spark and Conformal Prediction

Machine learning has been widely used in a variety of fields and many ML algorithms exist for predicting unseen examples based on classification, regression etc. both in supervised ML or for clustering techniques in unsupervised ML. However, the conventional ML algorithms lack information about the reliability or confidence of the predictions made for the new examples [54]. A common assumption is that a model will give predictions for future examples with performance comparable to that which it gave for initial test examples. However, a level of uncertainty obviously exists for the new observations which may differ from the test set – this has led to discussions and fuzzy definitions of a model’s “applicability domain”. In iterative virtual screening, while predicting ligands using QSAR models, a high
5.2. EFFICIENT ITERATIVE VIRTUAL SCREENING WITH APACHE SPARK AND CONFORMAL PREDICTION

level of confidence is important in the predictions for individual examples to enable informed decisions to be made. Conformal prediction can provide such confidence in predictions at a given level and also answer the question: “How good is our prediction?”.

Here we present a novel iterative, distributed, large-scale strategy for SBVS using Spark’s MLlib library, distributed conformal prediction \cite{68} and SVM \cite{78}. The aim is to avoid docking molecules that can be predicted to be “low-scoring” ligands with a certain level of confidence. To achieve this, we dock a subset of molecules iteratively and the conformal predictor is re-trained until the model reaches a certain efficiency level, after which all remaining ligands predicted as “high-scoring” are docked (see Figure 5.1). The strategy is different from earlier studies \cite{221, 223} that predicts a set of molecules to be docked in each iteration and improve the set of molecules in each iteration, whereas here we predict and filter out only “low-scoring” molecules. Comparing to the SBVS (Paper II), we show that CPVS is able to reduce the number of docked molecules by 62.61% against 4 different targets, while retaining an accuracy for the top 30 hits of 94% on average and gaining a speed-up by a factor of 3.7. The method is available as an open-source repository on Github named spark-cpvs \cite{18}.

5.2.1 Methods

5.2.1.1 Data

The same SureChEMBL dataset employed in the SBVS (see Section 4.3.2.2) was utilized in the CPVS study. The molecules were described using a Spark-based parallel implementation \cite{69} of the signature molecular descriptor \cite{99} and the consecutive signature heights were set at 1-3, i.e., an atom at a distance of at most 3 edges. An earlier study \cite{46} suggests that the signature height 1-3 produces good results for molecular classification with SVM-based models. OEDocking TK \cite{21} was used as the docking software and HIV-1 protease \cite{52}, PTPN22, MMP13 and CTDSP1 \cite{149} were chosen as the target proteins for docking.

5.2.1.2 CPVS Iterative Strategy

The objective of CPVS is to reduce the total computation time by avoiding the docking of molecules that are predicted to be “low-scoring” ligands and by only docking the compounds that are predicted to be “high-scoring” ligands with a certain level of confidence. The CPVS iterative strategy workflow is shown in Figure 5.2.

Initially, signatures were computed for all the molecules in the whole dataset, and two copies of the dataset were made: $Ds$ and $DsComplete$. An initial sample of $DsInit$ number of molecules was randomly taken from $Ds$ and docked against a chosen receptor and scores were calculated. To form a training set, docking scores were converted to class labels \{0\} and \{1\}, representing “low-scoring” and “high-scoring” ligands respectively. This was done using a 10-bin histogram of
the docking scores where labels were assigned to molecules in different bins. A
conformal predictor was trained on the training set and predictions were made on
the whole dataset, $Ds_{Complete}$. The molecules were classified as “low-scoring”
ligands $\{0\}$, “high-scoring” ligands $\{1\}$ and “unknown”, i.e., both labels $\{0, 1\}$
or empty $\{}$. The predicted “low-scoring” ligands were removed from $Ds$ in each
iteration and hence were never docked. The efficiency of the model was computed
by finding the ratio of single label predictions $[216]$, i.e., $\{0\}$ and $\{1\}$ against all
predictions. The process was then repeated iteratively with a smaller data sample
$Ds_{Incr}$ from $Ds$. The predictor was re-trained until it reached an acceptable level
of efficiency, and all the remaining “high-scoring” ligands were docked. The scores
of all the docked molecules were sorted and the accuracy for the top 30 molecules
was computed against the results from SBVS (Paper II) where all the molecules
were docked $[153]$.

![Figure 5.2: CPVS Iterative Strategy Workflow](image)

5.2.1.3 Modelling

We employed a Mondrian inductive conformal prediction (ICP) technique and SVM
as the underlying modelling method. SVM has been widely-used for predictive
modelling $[215]$, $[163]$. We used linear SVM, which has previously shown positive
results for QSAR modelling $[168]$, $[202]$, and its Spark MLlib-based implementation
with L-BFGS optimization because it performs well with imbalanced datasets. A
maximum of 50 iterations were used for the L-BFGS optimization. The training
set was randomly divided into two parts consisting of 10% of the original set as a
calibration set and the remaining 90% as a proper training set, and the confidence
level was set at 80%, which has been shown to work well in earlier studies $[169]$ with imbalanced datasets.
5.2. **EFFICIENT ITERATIVE VIRTUAL SCREENING WITH APACHE SPARK AND CONFORMAL PREDICTION**

5.2.2 Results

Improving the strategy means that a number of parameters need to be selected in order to reduce the overall time for the virtual screening. This includes minimizing the total number of docked molecules and keeping the size of training sets used for modelling as small as possible to avoid overly time-consuming training.

5.2.2.1 Initial Training Set and Labeling Strategy

The first predictive model in the analysis is of critical importance, and thus the initial training set must be large enough to produce robust results with a minimum number of false positives. The sizes for the initial training set $D_{sInit}$ that were tested were 50K, 100K, 200K and 300K.

Docking scores were divided into 10-bin histograms, where the bins were assigned as either “low-scoring” or “high-scoring”. Four combinations were evaluated: 1\_6, 1\_5, 1\_4, 2\_4, where the first number is the highest bin for “low-scoring” ligands and the second number is lowest bin for “high-scoring” ligands. For example, the combination 2\_4 means that bins 1 and 2 contain the “low-scoring” ligands while bins 4 through 10 contain the “high-scoring” ligands. The unassigned bin 3 is excluded from the training process. Figure 5.3 shows an example of a docking score histogram for a sample of 200K ligands in log scale. The data distribution was skewed because we had fewer molecules with high scores, which is normal for these types of datasets as only a few ligands have a good fit with the target protein and the majority will not bind with high affinity. One could sort the dataset to get the top and bottom scores and label them, but due to data shuffling in the distributed sort, that could be expensive. We expand on this in the discussion Section 5.2.3.

The labeling of the initial sample of $D_{sInit}$ as “low-scoring” ligands needs to contain as few (observed) good binders as possible, and hence the number of bins selected as class 0 should be kept low. The labeling of “high-scoring” ligands should minimize the chance of not including (observed) good binders, and therefore the number of bins selected as class 1 should be kept high. This reasoning formed the basis for choosing the bin combinations that were evaluated (see Table 5.1).

Table 5.1 shows the effect of different combinations of the $D_{sInit}$ size and labeling parameters on accuracy and efficiency after the first iteration. Each run was repeated 10 times, and the average and standard deviation for the accuracy and the efficiency were computed. In general, both increased efficiency and accuracy were reported with the increased size of $D_{sInit}$, but the labeling strategy based on the bin combinations also affected the results. The runs with $D_{sInit}$ size 50K and 100K were discarded because of the risk of fluctuation in the first model due to sampling issues with smaller datasets, which are observable by higher variance in the accuracy. For the remaining runs, the best combination of high model accuracy and efficiency was sought. Higher accuracy of the initial model reduces the chances of discarding actual good binders, and higher efficiency of the initial model implies fewer iterations to reach sufficient model efficiency in the iterative model building.
We selected run 10 in Table 5.1, i.e., the parameters with $DsInit$ size 200K and bins 1_5, which had a mean accuracy of 96.34% and an efficiency of 76%.

Table 5.1: Effect of $DsInit$ size and bin combination on Accuracy and Efficiency for the initial trained model (repeated 10 times).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50K</td>
<td>1_6</td>
<td>45.33</td>
<td>47.22</td>
<td>65</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>50K</td>
<td>1_5</td>
<td>65.33</td>
<td>43.95</td>
<td>63</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>50K</td>
<td>1_4</td>
<td>78.34</td>
<td>41.31</td>
<td>44</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>50K</td>
<td>2_4</td>
<td>94.34</td>
<td>4.46</td>
<td>79</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>100K</td>
<td>1_6</td>
<td>89.67</td>
<td>6.37</td>
<td>73</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>100K</td>
<td>1_5</td>
<td>94.67</td>
<td>5.92</td>
<td>75</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>100K</td>
<td>1_4</td>
<td>88.34</td>
<td>29.91</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>100K</td>
<td>2_4</td>
<td>89.67</td>
<td>7.45</td>
<td>91</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>200K</td>
<td>1_6</td>
<td>93.00</td>
<td>3.99</td>
<td>65</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>200K</td>
<td>1_5</td>
<td>96.34</td>
<td>1.89</td>
<td>76</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>200K</td>
<td>1_4</td>
<td>97.67</td>
<td>2.25</td>
<td>43</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>200K</td>
<td>2_4</td>
<td>90.34</td>
<td>9.74</td>
<td>91</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>300K</td>
<td>1_6</td>
<td>86.67</td>
<td>8.01</td>
<td>44</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>300K</td>
<td>1_5</td>
<td>95.34</td>
<td>4.50</td>
<td>63</td>
<td>17</td>
</tr>
<tr>
<td>15</td>
<td>300K</td>
<td>1_4</td>
<td>98.34</td>
<td>1.76</td>
<td>54</td>
<td>22</td>
</tr>
<tr>
<td>16</td>
<td>300K</td>
<td>2_4</td>
<td>86.00</td>
<td>7.17</td>
<td>94</td>
<td>5</td>
</tr>
</tbody>
</table>
5.2. EFFICIENT ITERATIVE VIRTUAL SCREENING WITH APACHE SPARK AND CONFORMAL PREDICTION

5.2.2.2 Incremental Model Building

In order to improve the efficiency of the model, each iteration needs a sufficient amount of new data added to the training set. The effect of the size of $DsIncr$ on the accuracy and the model efficiency is shown in table 5.2. We evaluated values 50K, 100K and 200K for $DsIncr$ and ran the iterative implementation until the desired efficiency was achieved. Each run was performed 20 times. The accuracy and the efficiency of the final models in all three cases were good and were similar to each other. In terms of time consumption, a $DsIncr$ size of 100K required the least total time to complete. The two major factors that contribute to the total time are the total number of docked molecules and the time used for model training and predictions. The numbers of molecules docked for all three settings were fairly similar, i.e., ~0.8 million. In all three settings, the model eventually reached the required 80% efficiency though the case with the smallest $DsIncr$ required more iterations. With the $DsIncr$ size as 50K, an average of 3.90 models needed to be trained, whereas with a $DsIncr$ size of 200K, although we need to train only 3.15 models on average, each model training takes more time because of the larger amount of data. Based on this observation, the size of $DsIncr$ was set to 100K for the final runs.

Table 5.2: Selecting $DsIncr$ size for incremental model building (repeated 20 times, mean values reported). Parameters $DsInit$ size = 200K and Bins = 1_5 for all runs. Time was calculated relative to 50K.

<table>
<thead>
<tr>
<th>$DsIncr$</th>
<th>Iterations</th>
<th>Accu.</th>
<th>Eff.</th>
<th>Docked mols (millions)</th>
<th>Total time (relative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50K</td>
<td>3.9</td>
<td>96.5</td>
<td>0.91</td>
<td>0.77</td>
<td>1</td>
</tr>
<tr>
<td>100K</td>
<td>3.35</td>
<td>96.84</td>
<td>0.91</td>
<td>0.81</td>
<td>0.96</td>
</tr>
<tr>
<td>200K</td>
<td>3.15</td>
<td>97.17</td>
<td>0.91</td>
<td>0.79</td>
<td>1.12</td>
</tr>
</tbody>
</table>

5.2.2.3 Efficiency of CPVS

We checked the performance of CPVS in terms of the total time reduction by comparing to our earlier work in SBVS (Paper II) [153] where the same dataset was processed with the same parallel environment but without the iterative strategy that includes ML component to filter out “low-scoring” ligands. The iterative strategy including the ML component was the key factor enabling computational efficiency.

• Experimental Environment -

We launched a standalone Spark cluster, along with HDFS on the SNIC Science Cloud (SSC) [24] using SparkNow [27]. SparkNow provides automated image creation and can initiate required services on virtual machines. A set of 12 nodes were launched each with 8 virtual CPUs (vCPUs), 16 GB of RAM,
CHAPTER 5. EFFICIENT INTELLIGENT MODELLING FOR VIRTUAL SCREENING AND APPLICATIONS

160 GB of disk storage and 40 GB of block storage. It was a completely virtualized environment and resembled a commodity computing-based cluster environment. One of the nodes was employed as the Spark driver, which did not take part in processing. The remaining 11 nodes were used as workers with a total of 88 cores.

- **Benchmarking** -
  As shown in Figure 5.4, both the SBVS and CPVS were executed in the same environment and the job completion time was recorded. SBVS, performing parallel docking, was executed once for each receptor and took 11.8, 8.30, 8.20 and 9.30 hours to complete against the HIV-1, PTPN22, MMP13 and CTDSP1 receptors respectively.

  CPVS was executed 10 times for each target receptor and the results are given in Table 5.3. The CPVS completed at least three times faster than SBVS for all four of the receptors and the accuracy was at least 90%. The average accuracy for all the four receptors was ~94%. Overall, the standard deviation for accuracy was low i.e. 4.33, which exhibits consistency in results. The average speed-up (SBVS total time / CPVS total time) for all four receptors was computed to 3.7.

![Benchmarking CPVS](image)

**Figure 5.4: Benchmarking CPVS**

### 5.2.3 Discussion

The docking step in structure-based virtual screening makes it a compute-intensive task that requires high-performance clusters or a cloud computing infrastructure
5.2. EFFICIENT ITERATIVE VIRTUAL SCREENING WITH APACHE SPARK AND CONFORMAL PREDICTION

Table 5.3: Results of the CPVS method for a set of target receptors. Results were averaged over 10 runs for each receptor.

<table>
<thead>
<tr>
<th>Receptor</th>
<th>Iterations</th>
<th>Accu. (percent)</th>
<th>Docked mols</th>
<th>Time (hours)</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIV-1</td>
<td>3.9</td>
<td>97.33</td>
<td>37.15</td>
<td>4.03</td>
<td>2.93</td>
</tr>
<tr>
<td>PTPN22</td>
<td>4.7</td>
<td>98.34</td>
<td>44.77</td>
<td>2.48</td>
<td>3.35</td>
</tr>
<tr>
<td>MMP13</td>
<td>3.5</td>
<td>89.00</td>
<td>33.34</td>
<td>2.10</td>
<td>3.90</td>
</tr>
<tr>
<td>CTDSP1</td>
<td>3.6</td>
<td>92.67</td>
<td>34.29</td>
<td>2.03</td>
<td>4.58</td>
</tr>
</tbody>
</table>

for timely completion. Our iterative CPVS strategy to filter out molecules from the actual docking shows positive results in that, on average, only 37.39% of the ligands were docked to reach an accuracy level of $\sim 94\%$ based on the top 30 binders, which saved about two-thirds of the total computation time. The results complement the previous work by Svensson et al. [205] who showed that 57% of the active compounds could be found by only docking 9.4% of the compounds using the DUD ligand set of 2,950 compounds with a conformal prediction approach. In CPVS, we employed a more realistic screening dataset of over 2.2 M compounds; the stepwise iterative docking and ML on such a large dataset was facilitated by the use of Spark for distributed computations and would have been complex and inefficient to carry out without a distributed data framework.

In a distributed environment, some of the data manipulation steps can be quite expensive as they could lead to a lot of data shuffling among the nodes. Some examples of methods in Spark involving shuffling across partitions are `groupByKey()`, `join()` or `sortByKey()`. The histogram approach was used during the labeling procedure to tackle shuffling caused by the distributed sort. We could have computed the top and the bottom percentiles but that would have included initial sorting of the data based on scores (which is an expensive operation to perform in a distributed environment). Hence, a lighter histogram method was employed, which also showed positive results.

Although the major gain of CPVS is the shortened virtual screening execution time, it also opens up possibilities for large-scale studies involving numerous target receptors and multiple large molecule libraries. Performing such parallel analyses on HPC and cloud resources means that the only significant limitations are resources and/or costs. Earlier, the on-demand orchestration of Spark clusters was a complex process, but nowadays it is a simple operation when working with the major cloud providers, and frameworks exist that greatly simplify this process both on private clouds (e.g., SparkNow[27]) and in HPC environments (e.g., spark-on-slurm[25], sparkhpc[26]).

Processing large datasets is usually time consuming and costly which means that it almost always requires large compute infrastructures to be able to complete jobs within a reasonable time frame. This restriction limited our opportunities for performing parameter sweeps in the study and necessitated a more tailored
approach. We also note that our results depend on the docking time, and hence the docking implementation (OEDocking TK in our case). However, we do not believe that major changes to parameters will be required in order to reach an efficient iterative docking with ML for other docking toolkits.

In this section we have presented CPVS, a novel iterative method for decreasing the overall processing time of virtual screening utilizing conformal prediction-based ML models and implemented using Spark, a next-generation big data framework. In Section 5.3 we present Paper IV PTPAAS, a web-based docking profile as a service where we develop models for multiple receptors using our CPVS strategy and provide predicted target profiles for multiple compounds against these ML models.

5.3 Predicting Target Profiles As A Service Using Docking Scores

Understanding side effects of drugs is extremely important while designing new drugs, especially in the initial stages of the drug discovery process. One way to analyze the side effects of drugs is to utilize ready-made target profiles. Building such target profiles through known in vitro drug-target bindings has been widely studied, however, another interesting technique is to create in silico target profiles for ligands [72], which helps in understanding off-target effects, as well as providing a new approach for predicting the binding affinity of Novel Chemical Entities (NCEs) against a number of targets.

In both predicting target profile modelling approaches, i.e. ligand-target interaction-based QSAR models and docking scores-based models, an important drawback is the lack of confidence information about the predictions. It is critical to have confidence in predictions as off-target drug reactions can directly affect human health. Another limitation in some of the previous studies is the lack of open-source code and the lack of extensibility options for the web services that were used.

Here we present an open-source, expandable web service (see Figure 5.5) to predict target profiles with confidence as a service (PTPAAS) for a panel of 7
5.3. PREDICTING TARGET PROFILES AS A SERVICE USING DOCKING SCORES

Figure 5.6: Vision: A predictive model for each of the receptor deployed in a cloud environment

targets where models are trained on QuickVina in silico docking scores, rather than on actual interactions in interaction databases. One objective of PTPAAS is to verify that docking scores can be used to build useful predictions for target profiles. The PTPAAS service also offers the functionality for docking chemical structures to a panel of targets on an individual compound basis using QVina. The PTPAAS service was deployed on a microservice framework, OpenShift Origin with Docker containers. Figure 5.6 represents the vision of this work i.e. all targets would have Docker containers along with the predictive models. All the Docker containers could be fired up in a Cloud environment and a compound of interest would be tested for all the targets to create a target profile for the compound. We used our earlier work, an iterative conformal prediction-based virtual screening (CPVS) strategy (Paper III) [40], to build the machine learning (ML) models. Models based on CPVS are ML models that predict at a given confidence level. The in silico approach is economically more practical, and the scores are also found in what is, relatively speaking, a shorter time for large molecular libraries. Molecular docking and ML models have been validated by docking well-known inhibitors with QuickVina for each of the 7 targets, calculating model efficiency and comparing model predictions with molecular docking scores.

5.3.1 Methods

5.3.1.1 Data and Tools

We used the clean drug-like molecule library, which was downloaded in ready-to-dock SDF format from ZINC [125]. Two separate datasets of ~2.3 M molecules and
200 K molecules were randomly sampled as the CPVS set and the validation set respectively from the clean drug-like molecule library. The CPVS set was used for modelling and internal testing whereas, for external testing, the validation set was used. The molecules were described using the signature molecular descriptor in the same way as in the CPVS and SBVS studies. Autodock Vina’s fast version, namely QuickVina 2, was used as the underlying docking software and the 7 target classes (see Table 5.4) were chosen from the proposed targets.

Table 5.4: Selection of Receptors: The Table represents the selected receptors and how they were selected. All the selected receptors must have a resolution of 2.5 (Å) or under and an RMSD of 2.0 (Å) or under.

<table>
<thead>
<tr>
<th>Target Class</th>
<th>PDB Entry</th>
<th>Resolution (Å)</th>
<th>RMSD (Å)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIV RT</td>
<td>1RT2</td>
<td>2.5</td>
<td>0.46</td>
</tr>
<tr>
<td>Acetylcholinesterase</td>
<td>1E66</td>
<td>2.1</td>
<td>0.34</td>
</tr>
<tr>
<td>HCK Tyrosine kinase</td>
<td>1QCF</td>
<td>2</td>
<td>0.29</td>
</tr>
<tr>
<td>Estrogen receptor</td>
<td>3ERD</td>
<td>2.03</td>
<td>0.57</td>
</tr>
<tr>
<td>Cyclooxygenase-2</td>
<td>3LN1</td>
<td>2.4</td>
<td>0.27</td>
</tr>
<tr>
<td>Carbonic anhydrase 2</td>
<td>1BNU</td>
<td>2.15</td>
<td>1.21</td>
</tr>
<tr>
<td>Purine nucleoside phosphorylase</td>
<td>1B8O</td>
<td>1.5</td>
<td>0.37</td>
</tr>
</tbody>
</table>

For each target class, the receptor PDB entries were chosen based on high resolution, i.e. 2.5Å or better. Several receptors were downloaded from the sc-pdb database for the selected 7 target classes together with their binding site information and were prepared using OpenBabel. Next, the RMSD for each receptor set was computed by docking the corresponding ligand available in the receptor-ligand complex; an RMSD below 2.0Å is considered a successful docking. Table 5.4 describes the final set of receptors, their PDB codes, their resolutions and the RMSD for the corresponding ligands. For testing purposes, a set of well-known inhibitors was compiled for each of the receptors. In each set, the average number of inhibitors was ~50 with a minimum of 43 and a maximum of 60 inhibitors.

5.3.1.2 Docking Score Based Modelling

For modelling, a modified CPVS version was used, while QuickVina was used for docking. CPVS is a conformal prediction and SVM-based, efficient, parallel, iterative virtual screening method. QuickVina is an open-source piece of software, so everybody can use our web service freely. In QuickVina, a ligand with a lower score is typically thought to have a stronger affinity to a particular receptor, so the labeling strategy in CPVS has been modified accordingly, i.e. low-score ligands have been labeled as 1 (high affinity) and high-score ligands have been labeled as 0 (low affinity). A sample dataset has been docked and sorted by docking scores. For model training, the top 10% and the bottom 10% of the docked sorted molecules
were used. The remainder of the method was the same as in the original method used in CPVS (Paper III) \cite{40}. The model training was carried out iteratively till the model reached and maintained the planned efficiency of 80\% or higher. An average of $\sim$0.53 million ligands were docked against each of the seven receptors during modelling. Comparing these results to those mentioned in the studies (see Table \ref{tab:5.5} cited in section \ref{sec:3.1.3} our study had a much larger training set for modelling, i.e. $\sim$0.11 million ligands per receptor model on average.

<table>
<thead>
<tr>
<th>Study</th>
<th>Average Training Ligands Per receptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yu et al. \cite{234}</td>
<td>5415</td>
</tr>
<tr>
<td>TargetNet \cite{231}</td>
<td>175</td>
</tr>
<tr>
<td>Bender et al. \cite{57}</td>
<td>1432</td>
</tr>
<tr>
<td>TargetHunter \cite{220}</td>
<td>216.6</td>
</tr>
<tr>
<td>Polypatharmacology browser \cite{60}</td>
<td>33.5</td>
</tr>
<tr>
<td>LaBute et al. \cite{142}</td>
<td>906</td>
</tr>
<tr>
<td>Wallach et al. \cite{218}</td>
<td>1236</td>
</tr>
</tbody>
</table>

**5.3.1.3 Predicting Target Profile as a Web Service**

The method is accessible through a web service that can provide a predicted target profile for a set of chemical compounds.

- **Functionality** - The web service consists of two parts: prediction of target profiles based on CPVS models, and molecular docking against individual targets. The user needs to provide a list of compounds in SMILES \cite{225} format for the target profile prediction. As output, the user receives a predicted target profile against all the available receptors for each of the compounds that are mentioned. Figure \ref{fig:5.7} shows an example of the predicted target profiles for two compounds. A compound’s prediction may be either “low-scoring”, “high-scoring”, or unknown, respectively represented by green, red, or blue colors. A low-score prediction generally means high affinity and vice versa in accordance with QuickVina. An unknown prediction means that either the model has failed to recognize a class for the compound or that it is predicted that the compound is a member of both classes with the given confidence level. The p-values for the low-score and high-score classes are also available and can be seen by hovering over the predictions. Once the target profiles are ready, the user can select interesting compounds based on predictions and use the functionality of QVina molecular docking to dock them. A receptor file can also be added in PDBQT format if a user requires a new receptor to be part of the profile. A model is then prepared and added to the profile on request.
CHAPTER 5. EFFICIENT INTELLIGENT MODELLING FOR VIRTUAL SCREENING AND APPLICATIONS

5.3.2 Results

5.3.2.1 Virtual Screening Evaluation

We separately docked well-known inhibitors for each of the 7 receptors using QuickVina to validate the virtual screening process and calculated the enrichment factor for the inhibitors' docking scores versus the docking scores of the ligands docked during the modelling procedure. The enrichment factor is among the most frequently used metrics to measure virtual screening success. A higher enrichment factor means better virtual screening performance and vice versa. Figure 5.8 shows the results of the QuickVina-based CPVS docking enrichment for all 7 receptors. The black dashed line depicts ideal scores, the gray dotted line in the middle depicts the scores that would be expected from random ligands while the blue solid line depicts the...
5.3. PREDICTING TARGET PROFILES AS A SERVICE USING DOCKING SCORES

inhibitor scores. The results indicate good or satisfactory enrichment for most of the receptors.

![Figure 5.8: Internal Docking Enrichment](image)

top % of ranked database

We also calculated inhibitor docking enrichment versus docking scores of an external validation set that the CPVS algorithm did not see during the initial modelling. In figure 5.9, the docking enrichment can be seen as a solid blue line. The enrichment illustrates satisfactory results and was used as the basis for evaluating the predictions from the model.

5.3.2.2 Model Evaluation

Two methods were used to evaluate the CPVS models: (i) by comparing the docking and predicted enrichment on the external validation set, and (ii) by calculating the efficiency of the model.
CHAPTER 5. EFFICIENT INTELLIGENT MODELLING FOR VIRTUAL SCREENING AND APPLICATIONS

- **Predicted vs Docking Enrichment** - In figure 5.9, the predicted enrichment on the external validation set is shown as a red solid line. To perform predicted enrichment, first predictions were made using the CPVS models. In addition to the predictions, the CPVS models also provided the p-values of the inhibitors and the external validation set for being predicted as either a “low-scoring” or a “high-scoring” ligand. Next, the p-values were used to compute unary enrichment values with the following formula.

\[
\begin{align*}
    &\text{If } (P_{\text{low-scoring}} > P_{\text{high-scoring}}) \\
    &\quad P_{\text{low-scoring}} * (1 - P_{\text{high-scoring}}) \\
    &\text{else} \\
    &\quad - P_{\text{high-scoring}} * (1 - P_{\text{low-scoring}})
\end{align*}
\]

These values were used to evaluate the predicted enrichment of known inhibitors against the external validation set. When comparing the predicted enrichment (red solid line) to the docking enrichment (blue solid line), the results were encouraging for the majority of the receptors except for PDB-ID 1B8O.

The percentages of inhibitors observed in the top 10% and 20% of the predicted ligands were also computed and are shown in table 5.6. The average percentage of inhibitors found in the top 20% of the predicted ligands was high, i.e., 63% of inhibitors for all receptors. The average percentage of known inhibitors found in the top 10% of the predicted ligands was 46%. The PDB-ID 1B8O receptor was again an exception where only 11% of the inhibitors were found in the top 20% of the ligands predicted and none were found in the top 10%. The 1B8O structure was compared to the other receptors, but nothing obvious was found that would cause such a behavior. The docking performance varies among different receptors and, in the 1B8O case, not many inhibitors existed in the top ligands, (see figure 5.9) which could be one reason for the 1B8O low performance.

- **Efficiency** - The models were also assessed using a measure of efficiency. Efficiency refers to the percentage of ligands predicted as being “low-scoring” or “high-scoring”, i.e. **single predictions** from the predictions on the entire dataset. The efficiency measures of the 7 models used for target profile prediction are given in table 5.6. All the models achieved the intended efficiency of 80% or higher for both the CPVS set and the external validation set.

5.3.3 Discussion

Target profiles are used in the initial stage of developing drugs to indicate the likely off-target effects of drugs. Here we present a new way of building predicted
5.3. PREDICTING TARGET PROFILES AS A SERVICE USING DOCKING SCORES

Using QuickVina’s docking scores, we built conformal prediction-based machine learning models. Overall, these produced good results, validating the process by evaluating virtual screening and the models that were developed. Therefore, the key finding of the research described in this section is that, by using docking scores, it is possible to build efficient models for predicting target profiles.

While earlier studies with ligand-target binding predictions based on docking scores exist, to the best of our knowledge, there have not been any tools or web services available for predicting target profiles based on docking scores; the web services that were available made use of databases for interaction values. Our work shows a new way of using docking scores to predict target profiles and, in the future, it would be interesting to compare the two approaches and examine the efficiency of a mixed system combining both approaches.
Table 5.6: The table represents the model efficiency of predictions on the complete CPVS set (from which the training set was taken) and the external validation set. The last two columns represent the predicted enrichment factor for inhibitors, i.e., the percentage of inhibitors found in the top 10% and 20% of the database search.

<table>
<thead>
<tr>
<th>PDB Entry</th>
<th>Eff. on CPVS set (%)</th>
<th>Eff. on Ext. Val. set (%)</th>
<th>Inhibitors in top 10 (%)</th>
<th>Inhibitors in top 20 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1RT2</td>
<td>93</td>
<td>97</td>
<td>32</td>
<td>68</td>
</tr>
<tr>
<td>1E66</td>
<td>93</td>
<td>94</td>
<td>60</td>
<td>67</td>
</tr>
<tr>
<td>1QCF</td>
<td>86</td>
<td>93</td>
<td>65</td>
<td>73</td>
</tr>
<tr>
<td>3ERD</td>
<td>93</td>
<td>92</td>
<td>65</td>
<td>78</td>
</tr>
<tr>
<td>3LN1</td>
<td>98</td>
<td>98</td>
<td>50</td>
<td>68</td>
</tr>
<tr>
<td>1BNU</td>
<td>87</td>
<td>87</td>
<td>47</td>
<td>75</td>
</tr>
<tr>
<td>1B8O</td>
<td>94</td>
<td>94</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Average</td>
<td>92</td>
<td>94</td>
<td>46</td>
<td>63</td>
</tr>
</tbody>
</table>

In science, openness and accessibility are important, therefore, for docking purposes, we moved from OEDocking that used in the original CVPS method to open-source QuickVina in this work. QuickVina, however, is slower than OEDocking and therefore restricted us to constructing a limited number of models, particularly for large datasets. We plan to include more models in the 7 receptor model set in the future.

5.4 Summary

In this chapter, we presented an efficient, intelligent model for virtual screening and a web-based application for profiling predicted targets. Going back to research question Q2 given in Section 1.2, we wanted to find out how we could increase the efficiency of Big Data Analytics methods in the life sciences and provide intelligent tools to the end-user based on these efficient Big Data Analytics methods.

To this end, we contributed an improvement on our earlier SBVS study and provided an intelligent iterative CPVS approach for virtual screening. The key advancement that enables the performance of CPVS is the new iterative strategy and the use of conformal prediction in an iterative way. Our iterative approach makes it possible to filter out predicted “low-scoring” ligands and only dock predicted “high-scoring” ligands using conformal prediction and SVM-based ML models. The strategy is successful as it saved two thirds of the overall virtual screening time, yet retained an accuracy of 94% on average for the top 30 hits, and demonstrated a speed-up by a factor of 3.7. With this work, we showed that the efficiency of a life science methodology can be increased using next-generation Big Data Analytics frameworks and ML methods. Our CPVS-based ML modelling was successful and thus we used it for developing the models for the PTPAAS project, where we implement a web-based application for predicting target profiles against multiple
receptors using docking scores. The work shows how we can build life sciences tools for end-users based on Big Data Analytics methods.

In Chapter 4 and 5, we demonstrated novel methods and techniques for processing life sciences data in batch form. In Chapter 6, we will describe efficient analysis of real-time life sciences data as data streams and solve issues related to real-time Big Data Analytics in the life sciences.
Chapter 6

Analysis of Life Sciences Data in Real Time

This chapter is mainly about Paper V [39] - it discusses the analysis of real-time streams in life sciences problems. In particular, a case study of brain data analysis is employed to present the analysis of large datasets from the life sciences in real time.

6.1 Introduction

In some scenarios, it is necessary to process and analyse big data as soon as it is produced, i.e. in an online analysis setting. Many applications involving large volumes of real-time data exist in the life sciences field e.g. body area networks (BAN) where patients are monitored through number of wearable devices. The devices consist of numerous sensors to record information regarding the patient’s heart rate, body temperature etc. The big data characteristics of volume, velocity and variety cause certain challenges in the real time, for example, in an online learning scenario, it is usually not possible to fit large datasets in memory and it is difficult to train models in a limited amount of time (both training new models and adding new data to an existing model with negligible delay are challenges) [62].

This chapter focuses on online pattern analysis of real-time neural data. The EEG, a non-invasive, multichannel technology for recording the activity of the brain, is commonly employed for monitoring epileptic patients in neurophysiological clinics. Normally in the clinics, a scalp EEG recording can produce data at a sampling rate of about 2kHz. However, in experimental studies [181, 100], the number of channels that are employed increased from tens to hundreds. To get an idea of the extent of the data that is produced, a continuous EEG reading for a patient which employs 24 channels at 256 Hz will produce approximately 1GB of data per day. In fact, an increased number of channels and a higher sampling rate can produce much larger datasets, e.g. 500GB per day [229]. With such
characteristics, EEG processing becomes a compute-intensive and data-intensive task.

Processing and analysing an EEG stream in real time has many applications e.g. an alarm signal (such as a text message) can be sent in advance to a relative of the patient when a seizure is likely to occur. Another application could be analysing EEG data in the ambulance en route to hospital for patients with head injuries – that way the staff at the medical facility could be provided with better knowledge of the injury beforehand.

This chapter presents the contribution to this area of research that is made in this thesis: a lightweight algorithm for detecting seizures in real time from EEG data using Spark streaming [236], which is a component of the Spark Big Data Analytics stack for parallel processing of streaming data. The lightweight algorithm was used to classify epileptic seizures from non-seizure data for a particular patient in real time. Furthermore, the algorithm is also trained in real time. A novel feature of “top-k amplitude measure” for classification was also introduced that helps to reduce the volume of the data while retaining the discriminating ability of the classifier. Thus successful data sampling, and training a model, with newly produced data have been demonstrated to be possible with negligible delay when working with real-time EEG data streams. The work shows that EEG data can be processed and analysed as streams in real time using Big Data Analytics technologies, and this development opens up the opportunity to process such data in a clustered computing environment. This work shows the contributions in this thesis towards the research question Q3 given in Section 1.2 i.e. it has been shown that analysis of real-time data streams can be improved in a life sciences online learning problem.

The rest of this chapter is structured as follows. Section 6.2 discusses important characteristics of epileptic seizures. Section 6.3 describes the implementation of a method for parallel real-time seizure detection in EEG data. Section 6.4 discusses the experiments testing the implementation and the results from those experiments, and Section 6.5 summarizes the chapter.

6.2 Epileptic Seizures and Characteristics

Seizure activity in a brain can manifest itself as a high amplitude EEG activity. A typical EEG with a seizure is shown in Figure 6.1. With the onset of an epileptic seizure, a change in the rhythmic activity of the electrode pair difference FP1-F7, FP1-F3, FP2-F4 and FP2-F8 can be seen – this is marked by the red rectangle. Not all the changes recorded by the EEG originate from the brain. EEG activity might include electrical activity from other body parts (e.g., the heart), environmental interference or instrument noise. For example, Figure 6.2 shows a normal EEG with a facial muscle artifact and should not be confused with seizure activity.

Significant characteristics of epileptic seizures need to be considered while implementing a system for seizure detection [177, 172]. First, no common pattern or
6.3 Parallel Real Time Seizure Detection in Large EEG Data

The goal was to process the seizure and non-seizures EEG data in real time, and classify it in a binary manner, with negligible delay. A classifier for real-time seizure detection was built – it is an algorithm that both learns from, and classifies, the EEG signals in real time. The following subsections discuss the architecture, feature selection, algorithm workflow and implementation of this lightweight algorithm.

6.3.1 Tools And Architecture

The aim was to analyze EEG as real-time data streams and at a rapid rate, as such we needed a tool that can provide us these facilities. Section 2.2.1 discusses the advantages of Spark Streaming as a Big Data Analytics stack for real-time applications. In this work, an OpenStack-based private cloud environment
was created. OpenStack is an open-source tool that monitors all of the resources in a cloud environment i.e., computes, storage and networking. Using the OpenStack dashboard, it is simple to quickly launch virtual machines (VMs), attach storage volumes, block storage or object storage.

The application's architecture is given in Figure 6.3. Apache Kafka was employed to transfer data in real time to the Spark cluster. Apache Kafka is a partitioned, replicated and distributed message log processing system. The EEG data was available in CSV files. Kafka producers published the EEG data as messages to the Kafka cluster, which can buffer the messages. Spark acted as a Kafka
consumer and received the data messages from the Kafka cluster as data streams. Kafka made it possible to divide the data into partitions and send it to individual Spark workers in a distributed manner without forming a bottleneck at the Spark master.

6.3.2 Feature Selection

Since the EEG data is of a non-stationary nature, it was necessary to partition the data into small portions. The data cannot be divided up on the basis of physiological activity, thus a commonly used technique is to partition data into windows of two seconds each, and features are then extracted from each window. Based on the seizure characteristics discussed in Section 6.2, the features of Top-K Amplitude measure, multi-window measure and multichannel measure were selected for this method. Amplitude measure has been mentioned [93, 150] as a feature associated with good results in earlier studies. We further improved the amplitude measure and created a novel feature of top-K amplitude measure discussed in the following section. The multi-window measure helps to analyse the data stream in real time, and the multichannel measure is useful for measuring the seizure on all the channels over a segment of the EEG data.

6.3.2.1 Amplitude of Top-K values

We found that in the event of seizure, the signal’s amplitude is stronger than during the non-seizure activity. In fact, earlier studies [93, 150] had successfully employed amplitude measure (average amplitude of all values over a segment) for seizure detection, mathematically defined as:

$$Amp = \frac{1}{M} \sum_{n=1}^{M} |x(n)|$$

(6.1)

where M is the number of values in each window and x(n) are the values in the currently processed window.

It was also found that during the non-seizure activity, most of the values appear near the rest position (the position in which the wave would sit if there was no disturbance moving through it), whereas in the event of seizure, values commonly sit far away from the rest position. Hence, we computed amplitude only for the top-k frequent values.

Figure 6.4 compares the amplitude per window for all the values and the amplitude of the top 60 frequent values per window. The major advantage of the top-k amplitude over the normal amplitude measure is the reduction in data volume. Consequently, the rest of the operations were applied on a smaller dataset and thus saved time. Furthermore, the reduced volume of the data did not affect the discriminating characteristics of the dataset (so the reduction did not lose important information). A common data size reduction method for EEG data is
“downsampling” where only every $M^{th}$ sample is retained for further analysis. In downsampling method, we only take a random sample of EEG and does not take EEG seizure activity into consideration. In contrast, this method was based on the fact that values frequently appear away from the rest position during seizure activity. The average top-k amplitude for a window is defined as:

$$Amp_{top-k} = \frac{1}{M} \sum_{n=1}^{M} |x_{top-k}(n)| \times t$$

where $M$ is the number of top-k values in each window, $x_{top-k}(n)$ are the top-k values in the currently processed window and $t$ is the number of times each top-k value appears in the current window.

6.3.2.2 Multi-window Measure

Individual windows work well for observing a single pattern, but they are unable to recognise the full pattern or evolution of a pattern that can be useful in detecting seizures. Therefore, a multi-window measure that remembers the previous activity for a particular number of windows was employed.

The abnormal brain activity takes some time to evolve into a full seizure, and it is only after that happens that the activity can be designated as a seizure. Different numbers of windows ($W$) were checked and it was found that a larger value of $W$ results in less false positives, but increases the latency, whereas a smaller $W$ value
results in low latency with a high number of false positives. Hence, a medium value of \( W=3 \) was selected (with the aim of finding a sweet spot), which resulted in low latency and also produced the least number of false positives. Overall, the algorithm takes 6 seconds to recognize a seizure, with each window being of two seconds in length. Here, we want to clarify that no overlap existed between the two sets of three windows.

Figure 6.5: Workflow for lightweight Seizure Detection Algorithm.

### 6.3.2.3 Multichannel Measure

The combination of all the channels over a window forms what is known as the multichannel feature. The multichannel feature was utilized to identify patterns on multiple channels. Seizures can appear on multiple channels simultaneously, while some artefacts exist that appear on only one or two channels. Hence, using the multi-channel feature makes it possible to distinguish such artefacts from the rest of EEG data and actual seizures. This feature was utilized by counting the number of channels that show the seizure activity over a single window.

### 6.3.3 Algorithm Workflow and Implementation

The goal of the method was to learn a non-linear threshold to recognize epileptic seizures in EEG data. It included two steps, namely a learning phase and a testing phase. The algorithm structure is given in Figure 6.5.
The data from every patient needs to go through the learning phase once, so that the characteristics for seizures for each individual patient can be learned. The algorithm is based on the idea that “To find what is abnormal, we must know what is normal”. The steps involved in the lightweight seizure detection algorithm are shown in Figure 6.5. Once the Spark cluster received the data stream, it was partitioned into windows. Then $\text{Amp}_{\text{top}} - k$ was evaluated for the non-seizure activity for each window. The cumulative mean of $\text{Amp}_{\text{top}} - k$ (CMA) was then computed for the previous windows. Around one thousand previous windows were processed to learn the normal pattern for the CMA for a single patient. Initially, fluctuations were noted in the CMA patterns, which slowly stabilized and become fully stabilized by the time the thousandth window was reached.

Once the one thousand windows were processed, the algorithm started to test the new EEG data for seizures. During the test phase, the CMA continued to be updated, unless a seizure activity occurred. Limiting the CMA during any seizure activity keeps the CMA threshold unaffected. As mentioned before, during seizures the $\text{Amp}_{\text{top}} - k$ is far from the CMA. To enable the CMA to better discriminate between seizure and non-seizure activity, the CMA was amplified by multiplying it with a parameter, $\text{boost}$. It was observed that a large $\text{boost}$ value decreased the number of true positives, whereas a small $\text{boost}$ value increased the number of false positives. Different values were tested and a $\text{boost} = 2.7$ was chosen, which worked well for all the patients. Any value above the $\text{CMA}^* \text{boost}$ was treated as a candidate for being a seizure and was filtered out in the first step, as illustrated in Figure 6.6. In the second step, the multi-window measure feature was employed to count the number of consecutive windows and, if three or more consecutive windows with a $\text{Amp}_{\text{top}} - k$ above $\text{CMA}^* \text{boost}$ existed, they were considered for the final check. In the final step, the Multiple-channel measure feature was applied to check the pattern of abnormal activity on multiple channels. Abnormal activity on more than two channels was considered as a true seizure activity. Most of the artefacts were removed with these steps.

Every step shown in Figure 6.5 included one or more Spark parallel operations.
6.4. EXPERIMENTS AND RESULTS

The following code snippet shows the steps involved for computing $\text{Amp}_{\text{top}-k}$:

```scala
// Count the number of times a value appears in a window
val amplitudeTopK = windowedEEG
  .map(x => math.abs(math.round(x._2.toDouble)))
  .countByValue()

// Top 60 frequent values
.map(_.swap)
.transform( rdd => rdd.context
  .makeRDD(rdd.top(60),4))

// Finding numerator and denominator
.map(x => (x._2 * x._1, x._1))
.reduce((a, b) => (a._1 + b._1, a._2 + b._2))

// Amplitude of Top-K for Normal Data
.map(x => (x._1.toFloat/x._2))
```

After the EEG data stream was divided into windows, the frequency of a value in each window was computed with the `countByValue()` operation. The most frequent values were then filtered out using `top()`, evaluating the numerator and denominator with the Map and Reduce operations, and then $\text{Amp}_{\text{top}-k}$ was computed. The complete code for the implementation is available on GitHub named `RealTimeEEG`.

6.4 Experiments and Results

Experiments were conducted to assess the effectiveness of the parallel seizure detection algorithm and its implementation. Two performance metrics were assessed: seizure detection ability and Spark streaming performance. An OpenStack-based virtual private cluster was utilized for the experiments with specifications as follows: HP ProLiant DL165 G7, 2XAMD Opteron 6274, 64 GB RAM and 32 cores per node. The virtual machines were based on x86_64 bit architecture. One VM was employed as the master and 10 as worker nodes. The worker nodes had 8GB RAM and 4 virtual CPUs (vCPUs) for processing. The master node had 16GB RAM and 8 vCPUs. The master node required extra memory to retain metadata for Spark workers and HDFS data nodes. All the nodes had the Ubuntu OS installed on them.

6.4.1 EEG Dataset

The EEG data was downloaded from the freely available CHB-MIT [191] database. Originally, the data was available as hour-long records in European Data Format (EDF) and was converted to CSV files for the experiments. After conversion, the size of the complete dataset was 1.44TB. The EDF format is not directly supported by Spark, however a readymade class for EDFInputFormat [128] was found. Before
CHAPTER 6. ANALYSIS OF LIFE SCIENCES DATA IN REAL TIME

sampling, each 2 second window contained 512 data points. With a total of 10 data streams, there were 5120 data points in each window.

6.4.2 Seizure Detection

The two metrics commonly used for assessing seizure detection ability are sensitivity and specificity. Sensitivity is the percentage of true positives for detecting seizures during the test phase, whereas specificity is the relative number of false positives given by the algorithm during the last 24 hours when no seizures occurred. The experiments were performed for 10 patients with a total of 47 seizures marked by experts in the CHB-MIT dataset. The data consisted of continuous scalp EEG recordings lasting for 240 hours at 256 Hz sample rate with 16-bit resolution.

6.4.2.1 Sensitivity

The algorithm correctly detected 91% of the overall seizures. Some of the seizures that were extremely short in length and thus were not detected by the algorithm. For example, in the case of patient number 2, three seizures of 1 min 12 s, 1 min 11 s and 9 s long existed. The seizures longer than one minute were detected, whereas the 9 second long seizure was missed. Table 6.1 shows the overall sensitivity performance of the algorithm.

<table>
<thead>
<tr>
<th>No. of Patients</th>
<th>Actual Seizures Found</th>
<th>Missed</th>
<th>Sensitivity (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>47</td>
<td>43</td>
<td>4</td>
</tr>
</tbody>
</table>

Some other studies Shoeb et. al. and Kiranyaz et. al. \[193, 135\] that used the same dataset as a benchmark for seizure detection problem reports sensitivity of 91-96% and 86% respectively. Shoeb et. al. employed SVM whereas Kiranyaz used evolutionary classifiers for classification. Our method provides comparable performance with sensitivity of 91%.

6.4.2.2 Specificity

The number of false positives for each patient are shown in Figure 6.7. Overall, the number of false positives was almost negligible. Most of the false alarms appeared in recordings containing seizures, which verifies the algorithm works well and false positives were found only around the actual seizures. In patients 6 and 7, artefacts appeared on multiple channels and caused a higher number of false positives.

6.4.3 Spark Streaming Performance

Experiments were also carried out to assess the streaming performance of Spark.
6.4. EXPERIMENTS AND RESULTS

6.4.3.1 Scaling And Latency

In this experiment, the scalability and latency of Spark streaming was assessed. During the experiments, EEG data was processed for 20 minutes and the average time taken for processing each 2 s window was found. The same experiment was performed starting with 4 cores, and then repeated on a larger number of cores, adding 4 cores at a time.

In Figure 6.8 it can be seen that, with 4 cores, each window was processed in over 4 seconds. This is a substantial degree of latency for a 2 second long window. As the number of cores was increased, the processing time decreased almost linearly and then became constant at just under 2 seconds once 28 cores were reached; at
that point it was possible to process the EEG data in real time. The experiment confirms that Spark streaming scales well for EEG data with an increasing number of cores with low latency.

Although overall Spark streaming performs well with EEG data, some drawbacks were observed, such as an inability to make windows based on input-data-time-stamps and the fact that it was only possible to make time-duration-based windows which then results in a variable number of data records in each window in a distributed environment.

6.5 Summary

Here, we have presented a parallel lightweight method for epileptic seizure detection in large EEG data as real-time streams. Going back to Q3 given in Section 1.2, we wanted to know how we can improve the efficiency of real-time stream analysis in life sciences problems in an online learning setting?

To investigate this question, a case study of processing large volumes of EEG data was employed, and a light-weight algorithm for seizure detection was developed and presented. The significant contributions in this work include the architecture, workflow and Spark streaming implementation of the algorithm. In an experimental scenario, the lightweight algorithm was able to detect seizures in real time with low latency and with a good overall seizure detection rate. A new feature was presented: the “top-k amplitude measure” for data reduction without losing the discriminating characteristics of the data. It was observed that, although Spark streaming scales well and produces results in real time with low latency, it has some drawbacks such as offering limited options for creating windows in streaming data. Taking all of these into consideration, Spark streaming has been demonstrated as having potential for real-time EEG analysis. Overall, this work shows that it is possible to process real-time streaming life sciences data efficiently in an online scenario through the reduction of data size with the novel feature of the top-k amplitude measure. The method also shows a complete architecture and procedure for performing such a task.
Chapter 7

Discussions

In the previous chapters, we presented novel scalable methods for virtual screening and data analysis in the Life Sciences. In this chapter, we discuss the impact of these methods in a larger context.

7.1 Parallel Methods for Virtual Screening

With the parallel methods for virtual screening in Paper I, LBVS and Paper II, SBVS, we successfully showed how scalable machine learning can be used to solve large dataset problems with ready-made ML models. Although ML algorithms were available for a long time, it was unclear how to use them in conjunction with big data frameworks. We presented a way to adopt the algorithms to big data analytics frameworks, allowing them to exploit parallelism and thus we are able to provide results in a shorter amount of time.

Another advantage of a method with ready-made models was saving the time for training the models. Reusability is important as many scientists have developed models in the past and proven models with predictable results already exist. Reusing these models allows us to quickly fire them up and get predictions without going through the training phase. On a higher level, this impacts drug discovery research in a positive way, reducing the overall time of the drug discovery process, which is normally slow, long and expensive.

In order to exploit parallelism and scalability, parallel HPC environments provide the best platform, however, they are typically expensive and not easily accessible by e.g. small pharmaceutical companies. Instead, cloud platforms, such as offered by Amazon or Google, provide a readily available platform with pay-per-use and on-demand allocations such companies can exploit. In order to use them, the applications need to be virtualized, though, and we showed how MR-oriented and Big Data Analytics frameworks for life science problems can be used on a virtual environment. The infrastructure and the configurations for Big Data frameworks like Spark, Hadoop are already handled by public cloud providers and handy graph-
ical tools are provided to access them. Thus, being able to perform scalable virtual screening in cloud environments is an important advancement. By relieving researchers from tedious configuration and setup tasks, they can better concentrate on their core research. So now, organizations that want to perform virtual screening can use their favourite cloud and scale up the cloud resources on demand with the increasing size of molecular libraries. Additionally, they do not have to worry about up-front investments, hardware configuration and maintenance costs.

### 7.2 Intelligent Iterative Methods for Virtual Screening and applications

The impact of the CPVS project is twofold. First, the advantages come from scalability and usage of low-cost virtualized computing environments as we discussed in Section [7.1](#). In addition, we have developed a novel iterative strategy, significantly reducing the time needed for docking by reducing the number of molecules docked. Generally, in such studies, the biological activity of the molecules is utilized for training ML models. In this work, we provide an alternative by training ML models using docking scores of molecules. Furthermore, the method can also be utilized when the biological activity of the molecules is unknown.

A common strategy in an iterative virtual screening is to dock an initial set of molecules. On the basis of the docking scores, the next most active subset of molecules is selected and docked. The purpose is to improve the docking set in each iteration. Rather than selecting the active molecules, we improved the docking set by predicting and removing the most inactive molecules in each iteration from the main molecular dataset. Even then we were able to show good overall accuracy. Another advantage of the method is flexibility for adopting different docking software, e.g., we easily switched from the OEDocking toolkit to QuickVina for the PTPAAS project.

As mentioned before, ligands can also bind to unintended targets. The docking score based ML models in the PTPAAS project are useful for understanding the off-target effects of drugs. Not only that, if a molecule binds to an unintended target, it may also provide hints for designing drugs for the other target. In a larger context, these strategies can save time and cost in the drug discovery process. A significant reason of drug discovery being expensive is failures in the last steps of the process. By knowing the off-target effects in the initial stages of a drug discovery process, we can prevent these failures and save time and costs.

### 7.3 Analysis of Life Sciences Data in Real Time

In Paper V, real-time EEG analysis, we presented a lightweight algorithm for seizure detection in large EEG data. Detecting seizures in real-time has some practical value. During a seizure activity, alarms can be sent to relatives or friends of the vulnerable. The application can also be used to send calls to a medical staff in a
hospital to visit the patient. Detecting patterns in neural data as a real-time stream can also be useful for people with head injuries in a remote area or at a place at a larger distance from a hospital. The results can be sent to doctors in advance to have a feeling of the injury e.g. such systems can be used in ambulances. Using the cloud as a compute resource in this application provides the same benefits of scalability and on-demand allocation, deallocation as discussed in Section 7.1 and 7.2.

We could have used a well-known nonlinear ML classification algorithm for this particular problem, but generally most of the typical ML classification algorithms are complex and the training procedure for such algorithms takes a significant amount of time that is an obstacle to process data and provide results in real-time. Hence, we decided to build a custom-made lightweight algorithm. Although we presented an algorithm specifically for real-time seizure detection, our approach is general for anomaly detection and can be applied in other domains as well. Surely, most of the selected features would be different in other domains but we believe that the methodology of learning a threshold through our lightweight approach is still workable for other types of anomalies.

In addition, we presented a new feature “top-k amplitude measure” for seizure detection, which is essentially a new data reduction method. Instead of taking the amplitude of all the values appearing in a window into account, only the top most frequent values in a window are used. Data reduction is important with large and massive dataset problems especially if we can retain the important characteristics of the dataset. In our case study we were able to discriminate the seizure and normal data even after reducing the dataset size. Again, we believe that the new feature can be used in several domains, especially for the data reduction. The method is better than random sampling where no consideration is taken regarding the characteristics of the sampled dataset.
Chapter 8

Conclusions and Future Work

8.1 Conclusions

The main aim of the thesis was to investigate the applicability of the next-generation Big Data Analytics tools for the life sciences and to provide novel methods for the research community to utilize when it comes to processing and analyzing large datasets using the next-generation parallel frameworks. In order to achieve the goal of the thesis, there were three main research questions that needed to be answered.

- **Q1:** How can the next-generation Big Data Analytics methods be adopted in life sciences problems and provide ready-made solution for large-scale problems with better resource utilization and speed-up?

- **Q2:** How can the efficiency of Big Data Analytics methods in life sciences be further increased using machine learning methods and how can intelligent end user tools based on them be provided?

- **Q3:** How can the efficiency of real-time stream analysis in life sciences problems be improved in an online learning setting?

To answer these research questions, we made the following contributions in this thesis.

Firstly, in Chapter 4, we investigated Spark as a scalable tool for virtual screening. This was done through ligand-based virtual screening experiments. We were able to show that Spark and HDFS could be used together with ML algorithms, e.g. to distribute and analyze data with SVM in parallel, thus providing a satisfactory scaling behavior and efficiency by improving resource usage and speed-up. Commonly, SVM-based applications are implemented in MPI, however we presented MapReduce as an alternative way to perform virtual screening in a public cloud environment. We also contributed a novel parallel structure-based virtual screening pipeline that provides a ready-made tool for the cheminformatics community to
perform large-scale virtual screening efficiently. This contribution verifies the applicability of next-generation big analytics tools for life sciences problems and shows the usefulness of next-generation big data tools for creating ready-made solutions in the life sciences field.

Secondly, in [chapter 5] we contributed a novel iterative conformal prediction-based strategy, which is a further improved, intelligent, virtual screening method that provides results with a similar level of accuracy to those achieved by previous methods e.g. SBVS but in a much shorter time. The docking step in the virtual screening process is time-consuming due to the large size of the molecule libraries. Conformal prediction along with SVM allowed us to confidently filter out the poor candidates and thus only lead-like molecules were docked. On top of this, we also provided an extendable web service for predicted target profiles for NCEs, docking against a battery of targets where 3D structures are available. Here, the novelty is the use of docking scores rather ligand-target binding affinities from the binding databases to create the target profiles. This enables life sciences researchers to compute structure-based target predictions. These target predictions can be used to predict off-target effects, for example in the early stages in drug discovery projects. The service was implemented in a microservice framework Play2 and deployed using OpenShift provisioned on the SNIC Science Cloud (SSC). This contribution demonstrates that the efficiency of a life sciences methodology can be increased through novel iterative methodologies including ML methods and shows how we can build life sciences tools for the end users based on Big Data Analytics methods.

Thirdly, in [chapter 6] we presented a parallel lightweight method for epileptic seizure detection in large EEG data as real-time streams. In this contribution, we have provided methods for solving the challenges associated with fitting large datasets in memory and quickly adding the newly produced data to the online predictive model without any data loss. In an experimental scenario, our lightweight algorithm was able to detect seizures in real time with low latency and with a good overall seizure detection rate. Also, we have introduced a new feature, the “top-k amplitude measure”, for data reduction that solves the problem of saving large datasets in memory and adding new data to the model on the fly. On the basis of the results that were obtained, we believe that this method can be of benefit in clinics for epileptic patients. This work shows that it is possible to process real-time streaming life sciences data efficiently and also demonstrates a complete architecture and procedure suitable for performing such a task.

The contributions detailed here achieved the main goals of the research underlying this thesis. We were able to show the applicability of next-generation big data tools for the life sciences problems by providing the novel methods, the architectures and the software implementations along with several improvements. These novel methods and improvements included better parallelism through Spark with better resource usage, a parallel pipeline for virtual screening, an iterative method for virtual screening, a docking score based web service for developing predicted target profiles and a method for seizure detection, along with a new feature for seizure detection and data reduction. In doing so, we addressed the above men-
tioned research questions and also achieved the larger goal of facilitating the life sciences research community for performing scalable analysis on large datasets in an improved manner.

8.2 Future work

Life sciences is an interdisciplinary field of research and as such future advancement in the relevant technologies especially in computer science and in artificial intelligence will play a key role in the development of the field. In this thesis, we have shown that novel Big Data Analytics technologies can be adopted for life science problems and with new technologies appearing in future, similar efforts will be needed to exploit them in the Life Sciences. This will only be possible by well connected cross-disciplinary efforts and collaborations, as prototyped in our projects.

In this work, we presented in silico methods and improvements for screening of molecular libraries using Big Data Analytics technologies. Similar work is being done by many researchers around the world working in isolation. To better exploit the collective knowledge and experiences created by these researchers, we see a pressing need for a system where such virtual screening methods can be easily shared and accessed by users and researchers.

While in our work, we concentrated on issues beneficial to Life Science researchers and professionals, another interesting future direction could be Intelligent Life science services for the general public e.g. in the area of intelligent health services to patients at home. For services targeting large populations, scalable technologies as presented in this thesis will be essential.

Along with the above given future directions, some interesting avenues for future research and possibilities for developing improvements also arose from the work in this thesis – they are presented in chapter 4, chapter 5 and chapter 6. We summarize them in the following sections.

8.2.1 Improved Load Balancing in SBVS

In the SBVS study [153], we parallelized VS using Apache Spark. The molecules were randomly distributed to worker machines by Spark. We noticed that the docking time varied among the molecules, which could be due to various reasons (features), e.g. the size of each molecule and its affinity to the receptor. This means that the speed-up of SBVS could be further increased if we load balanced the molecules on the worker machines. So a further avenue of research would be to do that by predicting the docking time in advance using ML algorithms.

8.2.2 Optimized CPVS

One direction for future research in relation to CPVS [40] would consist of further improving the CPVS method. For example, because the molecule scores that are
used in CPVS are a range of values, regression modelling could be employed. In the initial development, we employed classification because regression modelling was not part of the Spark MLlib at the time of implementation. Another interesting avenue for improving CPVS would be to investigate different labelling strategies and their effects.

8.2.3 CPVS via Novel Machine Learning Methods

In the CPVS study, we employed Conformal Prediction and used SVM as the underlying algorithm. Theoretically, it is possible to use any ML algorithm along with CP. It would be interesting to see how other novel ML algorithms, e.g. deep learning, perform in a similar iterative setting. Deep learning has become a popular ML method and its use has already resulted in improvements in QSAR predictions [79].

8.2.4 Extending PTPAAS

In the PTPAAS study [37], we prepared docking profiles against a total of seven targets. We would like to extend PTPAAS with more targets for a more complete docking profile, and then compare/combine that with other ligand-based approaches based on interaction values from the ChEMBL and ExCAPE-DB databases.

8.2.5 Complete Pipeline for EEG Analysis

In the EEG study [39], we presented a method for real-time seizure detection with Spark. Looking to the future, a complete general analysis pipeline could be developed for EEG data, which could benefit patients with a wider range of medical conditions.
Bibliography


[233] J. Yoo, L. Yan, D. El-Damak, M.A.B. Altaf, A.H. Shoeb, and A.P. Chand- 
drakasan. An 8-channel scalable eeg acquisition soc with patient-specific 
seizure classification and recording processor. IEEE journal of solid-state 

[234] Hua Yu, Jianxin Chen, Xue Xu, Yan Li, Huihui Zhao, Yupeng Fang, Xiuxiu 
Li, Wei Zhou, Wei Wang, and Yonghua Wang. A systematic prediction of 
multiple drug-target interactions from chemical, genomic, and pharmacologi- 


An efficient and fault-tolerant model for stream processing on large clusters. 

[237] J. Zander, M. Hartenfeller, V. Hähnke, E. Proschak, S. Besier, T.A. Wichel- 
haus, and G. Schneider. Multistep virtual screening for rapid and effi- 
cient identification of non nucleoside bacterial thymidine kinase inhibitors. 

epileptic seizure detection in scalp eeg recordings using an algorithm based on 
wavelet packet transform. In IEEE Transactions on Biomedical Engineering, 

implementation for high-throughput virtual screening using autodock. Bmc 

[240] X. Zhang, S.E. Wong, and F.C. Lightstone. Toward fully automated high 
performance computing drug discovery: a massively parallel virtual screen- 
ing pipeline for docking and molecular mechanics/generalized born surface 
area rescoring to improve enrichment. Journal of chemical information and 

framework to implement molecular docking of large-scale virtual screening. 
In Services Computing Conference (APSCC), IEEE Asia-Pacific, pages 350– 
353, 2012.

[242] Zsolt Zsoldos, Darryl Reid, Aniko Simon, Sayyed Bashir Sadjad, and A Peter 
Johnson. ehits: a new fast, exhaustive flexible ligand docking system. Journal 