Master Thesis

Deep Learning Model for DNA Reads through Nanopores

by
Ángel Díaz Carral

Thesis submitted for the degree:

Master in Physics of Complex Systems

of the

NATIONAL DISTANCE EDUCATION UNIVERSITY, SPAIN

Supervisor: Dr. Elka Radoslavova Koroutcheva

in collaboration with

UNIVERSITÄT STUTTGART, GERMANY

Supervisor: Dr. Maria Fyta

Abstract

DNA molecules can electrophoretically be driven through a nanoscale opening in a material giving rise to measurable electronic current blockades. These signals can be used to detect the translocating molecules. Additional read-out protocols based on electronic current signals across the raw signals from the nanopore region are expected to also identify the DNA sequence, that is the order of the nucleobase identity along the molecules. Nevertheless, the relevant measurements often include many errors. In order to reduce these errors, increase the read-out fidelity, and interpret the experimental observations, a methodological approach based on unsupervised and supervised learning to interpret the DNA events is proposed.

In this work, experimental ionic traces from molybdenum disulfide nanopores threading DNA nucleotides are used to train an unsupervised Machine Learning model for identifying distinct molecular events through the 2D nanopore based on the ionic current blockade height and unrelated to the traditional dwell time for each DNA event. Within this approach, the blockade level information is implicitly included in the feature space analysis and does not need to be treated explicitly. It is possible to show the higher efficiency of the blockade height over the traditional dwell time also with regards to coping with sparse nanopore data sets. This approach allows for a deep insight into characteristic molecular features in 2D nanopores and provides a feedback mechanism to tune these materials and interpret the measured signals.

Afterwards, the aim of the supervised learning is to use experimental data for training a Neural Network model in order to improve the identification of different nucleotides threading the nanopore. Different training sets can be obtained from different solid-sate nanopores and experimental conditions. Using different neural network architectures such as DNN or CNN it is possible to take advantage of these different training sets and compute a NN algorithm that is capable of optimizing and accelerating the nanopore read-out.

Resumen

Las moléculas de ADN pueden moverse electroforéticamente a través de orificios en materiales a escala nanométrica, dando lugar a corrientes electrónicas medibles producidas por el bloqueo del orificio al pasar la molécula. Estas señales pueden ser utilizadas para detectar translocaciones del ADN. Se espera que los protocolos de lectura adicionales basados en señales de corriente electrónica sobre las señales sin procesar identifiquen la secuencia de ADN, que es el ordenamiento de las bases nitrogenadas a lo largo de las moléculas. Sin embargo, las mediciones de relevancia a menudo incluyen muchos errores. Para reducir estos errores, aumentar la fidelidad de lectura e interpretar las observaciones experimentales se propone un enfoque metodológico basado en el aprendizaje no supervisado y supervisado de cara a interpretar eficientemente los eventos de ADN.

En este trabajo se utilizan señales de corriente iónica experimentales, obtenidas de la translocación de nucleótidos de ADN a traves de nanoporos 2D de disulfuro de molibdeno, para entrenar un modelo de Machine Learning no supervisado con el objetivo de identificar distintos eventos moleculares. Este procedimiento no relacionado con el tiempo de permanencia tradicional para cada evento de ADN, sino con la altura de bloqueo de la corriente iónica. Dentro de este enfoque, la información de la corriente de bloqueo de los niveles que componen la señal completa de cada evento, entendido por evento la señal obtenida de cada translocación a traves del nanoporo 2D, está implícitamente incluida en el análisis del espacio de características y no necesita ser tratada explícitamente. Es posible mostrar la mayor eficiencia que tiene la altura de bloqueo como descriptor con respecto al tiempo de permanencia tradicional, incluso también en lo que respecta a hacer frente a los conjuntos de datos de nanoporos pequeño tamaño. Este enfoque permite una visión profunda de los característicos descriptores moleculares en nanoporos 2D y proporciona un mecanismo de retroalimentación para ajustar estos materiales e interpretar las señales medidas.

Posteriormente, el objetivo del aprendizaje supervisado es utilizar datos experimentales para entrenar un modelo de Redes Neuronales con el fin de mejorar la identificación de diferentes nucleótidos que pasan a través del nanoporo. Se pueden obtener diferentes conjuntos de entrenamiento a partir de distintos nanoporos de estado sólido y condiciones experimentales. Usando diferentes arquitecturas para Redes Neuronales como DNN o CNN, es posible aprovechar estos conjuntos de entrenamiento distintos y calcular un algoritmo NN que sea capaz de optimizar y acelerar la lectura mediante nanoporos.

Acknowledgements

I would like to express my special acknowledgements to my supervisor and leader in Germany Dr. Maria Fyta, you have played a vital role as a second mother and a mentor during my research stage at the Uni Stuttgart. Thanks for your advice, which has allowed me to grow as a research scientist.

I would also like to thank my tutor from Spain, Dr. Elka Radoslavova, for her great guidance, feedback and recommendations throughout the thesis. Thanks for your flexibility as professor and for being always willing to help me.

Thanks to my lovely family. I have no words to express my very profound gratitude for all of the emotional, financial and unfailing support that you have given me throughout these last years.

A.D.C., C.S.S., and M.F. thank I. Steinwart for an insightful discussion. ADC and MF acknowledge financial support from the EXC 310 SimTech Cluster of the University of Stuttgart. CSS and MF thank the Juniorprofessorenprogramm funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK) for financial support. A.R. and K.L. would like to acknowledge financial support from the Swiss National Science Foundation (Grant numbers: 200021.153653, BSCGI0.157802)

Contents

List of Figures

List of Tables

1 Introduction

This section presents the context, motivation and objectives of the project, as well as the state of the art. It finishes by explaining the thesis organization through an outline.

Ultra-fast and personalized genome sequencing is the state of the art in nanobiotechnology. New generation genome sequencing is based on "single molecule" translocation methods. One of the most promising ways to implement a technology aiming at ultra-fast low-cost sequencing of DNA is by using nanopores [1]. A nanopore is a hole with a nano-diameter in a thin membrane. When it is immersed in an electrolyte and a voltage is applied, the dissolved ions create an electric current through the pore that drags molecules. This is used to probe single molecules with the resistive pulse technique [2], a method for determining the increase in resistance in the nanopore. Nanometer-sized pores opened in materials are efficient single-molecule detectors that can electrophoretically thread biomolecules, such as DNA, RNA, and proteins to identify and distinguish molecular events, as well as single molecule properties [3]–[5]. Solid-state nanopores [6], [7] such as boron-nitride [8], silicon-nitride [9], MoS_2 [10], [11] and graphene [12]–[15] have played a valuable role in ultra-fast DNA sequencing [16]–[19]. Through a nanopore setup, precise information on the length and type of molecule, as well as on the characteristics of molecular events can be acquired [20]–[22]. This information is typically mapped onto ionic current blockades measured across the pore [23]. The duration of a current blockade denotes the translocation or dwell time for the event, i.e. the time required for the DNA translocating through the nanopore [24]. These signals can also be used to discriminate among different homopolymers [25] or different folding events of the molecule through the pore [26]. Sequencing, though, requires the detection not only of the DNA molecules as a whole, but that of their subunits, the nucleobases. For this, certain protocols have been proposed, which involve the interpretation of the longitudinal ionic current [27], the transverse tunneling current across the nanopore and perpendicular to the translocation direction [28] or optical measurements coming from the pore region [29] obtained by a capacitance-based read-out signal [30].

Although very promising, the distributions of the ionic current through a nanopore are often very broad and typically overlapped for different molecules or events. Dwell time cannot efficiently capture the rich dynamics of each translocation even given through the typical multi-level nature of the ionic current blockades. Molecular aspects are missing from the traditionally used dwell time. On top of this, thermal fluctuations, the surrounding salt solution, the applied electric field, the high flexibility of biomolecules, the nanopore-molecule interactions are some of the factors introducing additional errors in the measurements. These factors decrease the signal-to-noise ratio (SNR) in the measurements, thereby also the detection fidelity [31]. The SNR can be enhanced through a tuning of the experimental setup [32] or by post-pro- cessing of the ionic current measurements obtained through the nanopore [33]. The latter is more efficiently achieved through the use of Machine Learning (ML) algorithms. Training ML algorithms to interpret nanopore data is notably the state-of-the art in the field. One of the most promising nanopore devices in the market, the MinION from Oxford Nanopores [34], [35] is delivering the sequencing data processed through ML [36], [37] and can, as an example, efficiently identify the position and structure of a bacterial antibiotic resistance island [38]. ML techniques can play a vital role in processing and recovering the information in the nanopore with robust statistics by creating automated models. Segmentation techniques, such as the cumulative sum (CUSUM) [39] or Bayesian-statistic-based [40] algorithms dissect the ionic current into subunits or levels that can describe information related to the specific molecule configuration, as well as the topology of DNA molecules passing through nanopores [41]. For classification tasks, the number of descriptors/features [42] taken for feeding the ML scheme can be optimized and their quality can be improved with dimensionality reduction and tree-based algorithms [43], [44]. In such schemes, an analysis of a feature space, by typically considering features such as the dwell time or the mean current blockade, can point to the most probable DNA translocation paths [39], [45]. Overall, the ML schemes for nanopore involve supervised learning and labeling of the data. *De novo* clustering [46] is - to our knowledge - the only unsupervised learning method, which though focuses in accelerating the data processing.

On a time scale lower than the dwell time of a whole molecule, the search of an "appropriate" feature space for training a ML algorithm is quite complex. Towards this goal, different ML protocols [47]–[49] have been developed to study protein-protein [50]–[52] or DNA-protein[53] interactions through a proper feature extraction and classification. Similar approaches for base-calling predictions have been used to study the interactions between DNA molecules and electrodes[54] embedded in nanopores.[55] In addition, deep learning techniques such as Hidden-Markov-based[56]–[59] or neural networks[36], [60]–[62] algorithms applied to nanopore data have shown the potential to improve the detection accuracy and automatize the discrimination of nucleobases towards ultra-fast DNA sequencing. ML techniques in conjunction with nanopores mainly focus on either guiding the learning process for optimizing the feature space, thus the error rates or improving the algorithmic scaling of the data processing. Along the above lines it will be presented the analysis and classification of the various possible configurations for the single nucleotide translocations through MoS2 nanopores using a clustering algorithm and deep learning techniques.

1.1 Research Objectives

This thesis focus on different aspects, in order to fill in the gap in using ML techniques to gain insight onto the molecular features inherent in the nanopore data. The objectives of this work can be enumerated as follows:

- Discover a new very efficient feature to identify molecular events from nanopore reads is proposed. This novel feature will be the key in obtaining more information on the underlying Physics of the molecular translocation that is missing in the literature.

- Implement an Unsupervised Learning algorithm with clustering methods to act on the nanopore information without guidance, in order to classify nanopore data and extract topology information from clusters.

- Implement a Supervised Deep Learning algorithm through neural Network techniques in order to accelerate the sequencing process using the novel feature instead of the ionic current raw signal.

1.2 Thesis Organization

In this chapter a brief overview about Machine Learning methods applied to reads from nanopores is presented. The rest of the thesis is structured in two main applications. The first part about Unsupervised Learning analyze the raw signal of nucleotide translocations, selecting the best candidates for being features that give insight onto the physics and topology of the DNA and nanopores. The second part, classification of DNA reads using neural networks, takes advantage of the novel feature proposed in the clustering analysis.
This thesis introduce DNA sequencing through solid-sate nano-pores and potential Machine Learning applications. Biological background in chapter 2 is treated, giving an overview of DNA sequencing, biopores and solid-state nanopores.
Chapter 3 gives a brief explanation about Machine Learning methods applied in this work, such as unsupervised learning clustering algorithms and neural networks from classification algorithms.
Chapter 4 goes into the methodology and implementation of the project. Here DNA data collection and preprocessing are explained, as well as the clustering algorithm and NN chosen and its optimization.
Chapter 5 focuses on the clustering analysis, showing the relevance of the novel method that is proposed.
Chapter 6 describes the different classification run with different NN architectures using the features extracted in the preprocessing.

Chapter 7 concludes the thesis by giving the goals achieved and future lines relevant towards DNA sequencing through nanopores.

1.3 Published Work

This thesis has emerged from peer-reviewed journal article produced at the Uni Stuttgart as part of my responsibilities as research assistant, published in July 2019 by IOP Science, 2D Materials [63]. The Unsupervised Learning part from this project is included in this publication. Here we expand the results from the clustering analysis and add a second main part related to Supervised Learning with Neural Networks.

- Angel Diaz Carral et al 2019 2D Mater.**6** 045011.
  `https://doi.org/10.1088%2F2053-1583%2Fab2c38`.

2 Biological Background

This chapter introduces the field of DNA sequencing and the use of solid-state nanopores.

Here, a brief introduction on DNA molecule framework and DNA sequencing is presented. The last section deals with nanopores and DNA translocation through these. Nanopores have a strong technological application in the area of DNA sensing and sequencing. New generation ultra fast methods are candidates for decreasing the cost of genome sequencing.

2.1 DNA molecules

Deoxyribonucleic acid (DNA) is a biomolecule contained in the living cells of organisms and store the genetic information. In combination with histone protein DNA molecules form the nucleosomes. These are folded through higher order structures in order to build chromosomes. Each of the 23 human chromosomes contains a single DNA, which is a very long biomolecule, reaching 1.5 mm [64]. The genome information is coded in four different nucleobases in the ATCG alphabet. Each letter denotes one of the four nucleobases Adenine, Thymine, Cytosine and Guanine, shown in Fig. 2.1 . Each base bonds with a sugar group making the nucleosides and a phosphate group forming the nucleotides. The phosphate group is a strong acid, that is the reason structures formed by nucleotides are called nucleic acids. Sugar groups from the nucleotides connect, creating a single-stranded DNA (ssDNA). In 1953, James Watson and Francis Crick discovered the molecular structure of nucleic acids, a rough model of which is shown in Fig 2.2 a) [65]. They identified the base pairing mechanism based on X-ray diffraction images taken by Rosalind Franklin et al. [66], where Adenine and Thymine form stable base pairs with two hydrogen bonds and Guanine and Cytosine through three hydrogen bonds. Single-stranded DNA molecules can additionally link together in antiparallel directions to form a double-helix DNA which is much more stable than RNA or most of proteins. Information can be stored within the double helical structure of the DNA. In humans, DNA forms chromatin combining with histone proteins. This protects the genomic integrity and stabilises the DNA structure into a small compact volume [67]. DNA is an extremely stable and symmetric molecule because deoxyribonucleotides lack a reactive hydroxyl group 2' and antiparallel DNA strands form a helical structure. Double-helix

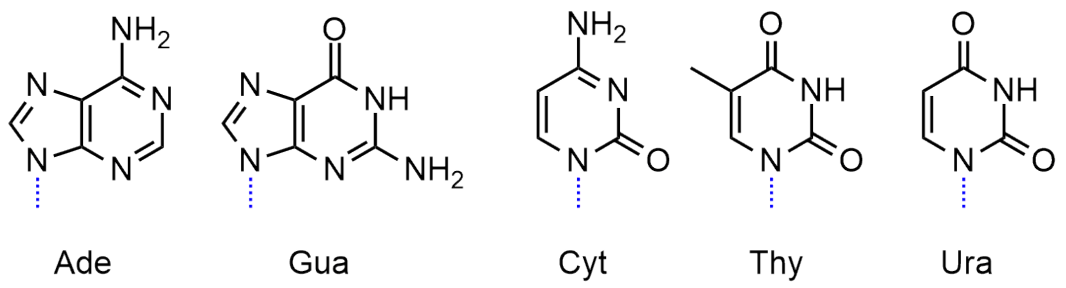

Figure 2.1: Five main nucleobases form nucleic acids [71]. Adenine, Guanine and Cytosine can build up both DNA and RNA, while Thymine only appear in DNA and Uracil in RNA molecules.

DNA is stabilized with hydrogen bonds, formed by bindings between purine and pyrimidine bases, and hydrophobic interactions between bases inside the helix. Both pairs of bases are complementary, thus every nucleotide in one strand is opposed to the matching partner on the other strand (Fig 2.2 b)). Nevertheless, this structural stability and regularity makes DNA molecules ineffective for catalysis processes [68]. This means DNA cannot catalyze cellular processes as catalytic RNA enzymes or ribozymes can. Many nanopore experiments use ssDNA, in order to detect the transport and access to genetic information instead of the double-stranded DNA (dsDNA) due to the higher stability and difficulty to decode genetics with the latter.

Each amino acid of a protein is specified by three RNA bases or codon. The genetic code is the set of base combinations which encode the 20 common amino acids of cells [64]. It is redundant, because 3 bases could specify 4x4x4 amino acids. As a result, three-nucleotide codons are more than enough to decode all them. All amino acids but methionine and tryptophan are encoded through more than a codon. Nonetheless, the genetic code is not ambiguous, because only one amino acid can be encoded by one codon. Most part of the genetic code contains instructions for making all the different proteins that the cell needs to function and to make more copies of itself. In cells, the information flows from the DNA to messenger RNA (mRNA) and finally synthesises proteins. A mRNA is similar to DNA in which the base Thymine is exchanged with Uracil (Fig 2.1) and the sugar is changed from a deoxyribose to a ribose [69]. Although, the functions and interactions of these molecules are not completely understood, the hypothesis of the mRNA is accepted. This assumption was used by F. Crick in order to build the central dogma of molecular biology [70], which simply states that DNA molecules encode RNA and RNA molecules encode proteins. According to this central dogma, the genotype of an organism is determined by the base sequence of its DNA, while its phenotype is a product of the proteins it procedures [68].

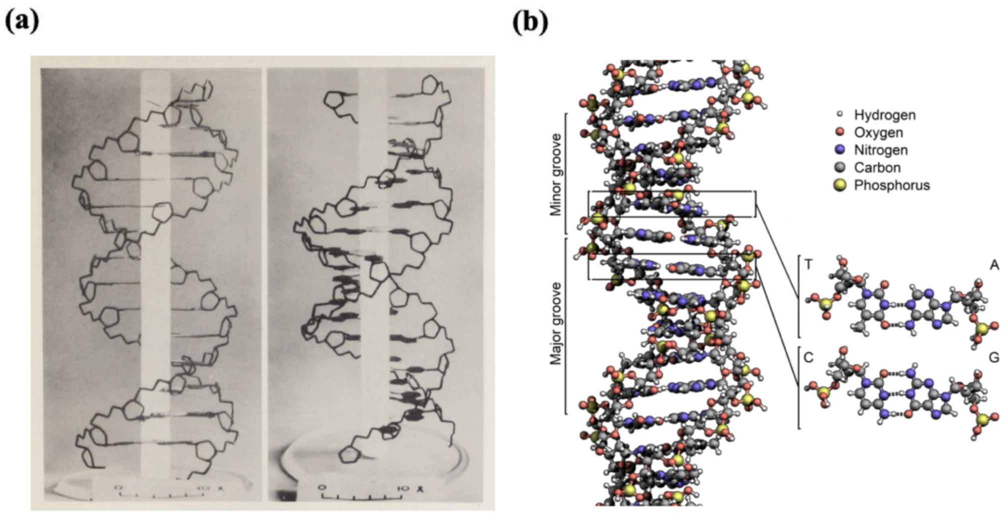

Figure 2.2: Double helical structure of DNA. Fig. 2.2 a) shows the first scale model from Watson and Crick [65]. Fig. 2.2 b) depicts the secondary structure of DNA showing in detail the structure of the four bases, Adenine, Cytosine, Guanine and Thymine, and the location of the major and minor groove [72].

2.2 DNA SEQUENCING

DNA sequencing determines the order of nucleotides in a molecule of DNA. Sequence of nucleotides of a genome underlies the protein complement of the cell and hence much of the phenotypic variation that we see between individuals. The development of methods for determining the sequence of nucleotides in DNA molecules have a great impact on human health and the evolution of life [73].

One of the first sequencing method previous to the famous Sanger sequencing was made by Allan Maxam and Walter Gilbert in 1976-1977, developing a DNA sequencing procedure that determined the nucleotide sequence of terminally labeled DNA molecule by chemical modifications [74]. According to A. Munshi [75], "The method requires radioactive labelling at one end and purification of the DNA fragment to be sequenced. Chemical treatment generates breaks at a small proportions of one or two of the four nucleotide based in each of four reactions (G,A+G, C, C+T). Thus a series of labelled fragments is generated from the radio labelled end to the first 'cut' site in each molecule. The fragments in the four reactions are arranged side by side in gel electrophoresis for size separation. In order to visualize the fragments, the gel is exposed to X-ray film for autoradiography, yielding a series of dark bands each corresponding to a radio-labelled DNA fragment, from this, the sequence may be inferred". In 1977, Frederick Sanger published a chain termination method based on introducing specially modified nucleotides, which allows DNA strands up to approximately 200 number of bases in order to give a reasonable basecalling accuracy [76]. The Sanger method was more efficient and used fewer toxic chemicals and lower amount of radioactivity than the method of Maxam and Gilbert. Sanger sequencing has been the technological reference of sequencing techniques. Although, during the last decade several next-generation method have turned into viable alternatives.

A handicap for first generation sequencing methods was the number of bases of a DNA molecule that they process, which is much smaller than the length of the genome for most of organisms. This limitation render sequencing genome completely a difficult problem. In order to resolve the whole genome sequencing problem, shotgun sequencing methods can be used. These techniques are based on the famous polymerase chain reaction (PCR) of K. Mullis, which amplifies the DNA samples size by clonation until obtaining multiple copies of the target genome. After applying PCR, these copies are randomly broken into small fragments or reads that are processed by the sequencing device. Fig. 2.3 depicts an example of the PRC applied to assembly sequencing. The key of the shotgun sequencing methods is the larger number of bases from DNA molecules that can be sequenced. The potential of Sanger sequencing methods has been improving over decades. Nonetheless, the cost of sequencing whole genomes remains too high. For this purpose, second generation sequencing methods were developed, in order to decrease the cost and sequencing time by reading lots of reactions in parallel of PCR segments or even a hole dsDNA. One of the most used second generation techniques is pyrosequencing. This method amplifies DNA, which first is isolated and ligated to adapters, in a water and oil emulsion, where takes places the PCR in order to clone the DNA molecules. Afterwards, luciferase is used to generate light after nucleobases are incorporated in a growing DNA chain [77]. Pyrosequencing decreases the cost per base and generates larger databases than Sanger sequencing.

The large size of DNA reads sequenced by PCR based methods needs computers in order to process the whole genome information. As DNA reads are obtained from lots of shorter and different sequences, it is necessary to assemble computationally every fragment to write a complete sequence. Gaps in the assembled sequence cane be filled by primer walking [75]. This technique is known as *de novo* assembly sequencing. For large genomes, it is possible to assemble millions of reads in a reasonable time by the use of computers [73]. Assembly sequencing methods have appeared as a new field due to the increasing interest on extracting genome information. On that respect, the Human Genome Project was set in 1988 in order to map and sequence the whole human genome [79]. The aim was to increase the accuracy of the DNA reads and at the same time reducing the cost. Different researching groups worked cooperatively focused on extracting the whole base pair sequence of the human. This information can provide potent means of organisms and contribute to help to biological analysis and the prevention of diseases [73].

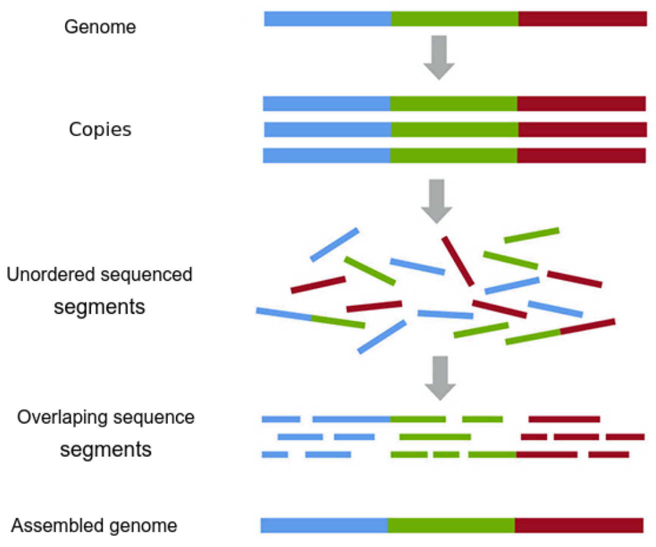

Figure 2.3: PRC technique applied to assembly sequencing processes [78].

Relevant insight into gene functions and genetic variation has been found over the last years with the sequencing of entire genomes. Cost efficient, high-throughput whole genome sequencing, which is going to provide the community with massive amounts of sequences, are the goals of new sequencing technologies that are rapidly evolving [80]. Decrease the cost of the genome sequencing is nowadays the objective of the "1.000 dollars genome" project [1]. This amount corresponds to the target cost for determining an individual's genome, being included in diagnostic tests. The scale of this ultimate goal is six orders of magnitude faster and three times cheaper than current state of the art technologies [1]. An ideal technology for this purpose would be a "single molecule" sequencing method, where the signal amplification is not necessary in order to identify DNA fragments, avoiding the addition of noisy signals that induces errors.

Sequencing field demands technological improvements on quality and faster reads. In this respect, third generation systems have the potential to greatly benefit *de novo* assembly methods. Third generation-sequencing methods have the potential to sequence nucleobases without chemical transformations. The technique called single molecule realtie sequencing (SMRT-seq) uses a processing polymerase to insert nucleotides opposite a template DNA strand [67]. This method can efficiently determine DNA nucleobases by its raw signal, which comes up from the movement of the molecules through nanopores. Chemical labelling can be used along with SMRT-seq, since bulkier bases have a more unique signature which can aid detection [81]. SMRT-seq provides multi-decoding techniques in one sequencing experiment, increasing the sequencing time. Current technology is still growing up and is only feasible for the sequencing of small genomes [67]. The boundaries of DNA size in sequencing is a limit for new technologies, because their power of resolving large DNA molecules is not enough in order to obtain information about individual bases. The high demand for low-cost, sequencing has driven the development of cheaper, portable, plug-and-play devices for high- throughput methods that parallelize the sequencing process, reading simultaneously million of DNA strands. High-throughput-based technologies are intended to lower the cost of DNA sequencing [75]. While new generation sequencer evolves, nanopore-based technologies have appeared as strong candidates for solving the low-cost sequencing problem [1]. Nanopore are based on the electronic detection of DNA sequence and have the potential of low sample preparation work, high speed, and low cost [3]. Future technologies along with computational power will enable to make progress in large genome sequencing and fields like psychology, ecology and climatology will benefit from this knowledge.

2.3 Nanopores and DNA tranlocations

A nanopore is a hole in a material with a diameter between 1 nm and 100 nm [82] used in new generation molecule translocation techniques. Fig 2.4 shows the schematic process of a translocation. There are two kinds of nanopores, the biological, like the protein nanopore used in the MinIon device [83] and the solid state nanopores. Solid state nanopores (silicon or based-substrate nanopores) can be made by drilling a hole into a membrane [84]. They exhibit relatively lower single molecule detection sensitivity com-

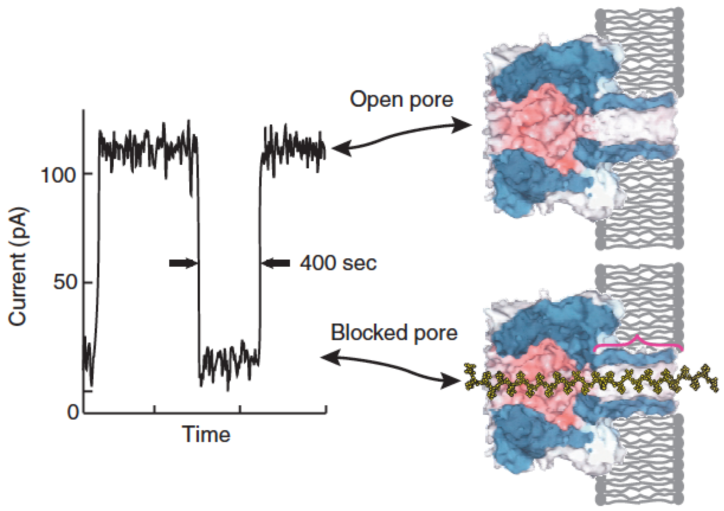

Figure 2.4: Ionic current blockades through nanopores. Traces of the ionic current amplitude through an alpha haemolysin nanopore clearly differentiate between an open pore and one blocked by a strand of DNA [3].

pared to biopores due to their intrinsic thickness and lack of control over surface charge distribution. The aim of the development of solid-sate nanopores is to avoid noise and overlapping when reading ionic current traces. For this purpose, the sensing region must offer spatial resolution enough for an efficient molecule sequencing. Ultra-thin membranes such as graphene or single-layer molybdenum disulphide can be use to build 2D nanopores for DNA sequencing [83]. Recently, MoS2 nanopores have shown the ability of differentiating DNA nucleotides with single-base resolution by slowing down translocation speed using an ionic liquid gradient across the nanopore [10]. Translocation time of a ssDNA through a MoS2 nanopore could be prolonged by reducing the proportion of molybdenum atoms to sulfur [85].

Nanopore sequencing is a very promising techniques that does not need the previous biotechnological methods of second-generation sequencers such as PCR or chemical transformations. In nanopore sequencing, DNA dispersed in an ionic solution is electrophoretically driven (due to its negative charge) through a nanoscale pore and being read. The nanopore is a highly confined region of space, in which the polynucleotides can be analyzed by various means. DNA molecules can be electrophoretically threaded through nanometer sized pores in a process named "DNA translocation" [1]. Once a controlled processing is enforced in the nanopore region, the native sequence of bases can be detected by the signal carried along. The time interval during which the current level remains decreased is proportional to the length of the molecule that blocks the nanopore, which give insights about translocation dynamics. Solid state nanopores are made of 2D materials.

3 Machine Learning

This chapter summarizes the machine learning methods used in this thesis, such as unsupervised learning clustering algorithm and neural networks.

Machine Learning (ML) involves algorithms used by computers for optimizing a performance criterion (e.g. character recognition) using example data or a past experience. It has emerged as a sub-field of Computer Science due to the massive collection of databases since the 80s related to engineering fields, speech and image analysis, pattern recognition or communications [86]. Learning algorithms are good candidates to substitute conventional way of extracting data from simulations, increasing the efficiency of the algorithm of interest. The goal of Machine Learning is to automatize computers for solving problems without the use of a program explicitly. This should be done through learning rules that save computational time and improve the accuracy of processes that humans cannot reach. Technologies such as image and voice recognition, personalized marketing or data analytics work along with Machine learning algorithms in order to learn and understand its insights [87].

In general, conventional learning algorithms involve main steps that build a pipeline. They start with the acquisition of data from the field of interest. Preprocessing and transformation of these data is important in order to be used as an input for the ML algorithm. Most relevant descriptors/features from the dataset are used to create a feature space, the place where the collected data lives. All these collected samples form the training set. The aim of the training step is to create a space where the dataset live and the computer must learn by splitting zones with the same features. If the learning process is supervised [88], then each sample belong to a class, that can be categorical or numerical. The set of classes from each sample represents the labels, that are the already known solutions. The next part is the optimization, where the computer must find coincidences between samples and, if labels are known, optimize the geometrical object that divides those zones. The goal of a ML algorithm is to optimize the splitting function of the feature space or find a new one in order to create a rule decision that can automatize the task. Fig 3.1 shows the scheme of a ML methodology including the domain knowledge that can dictate the choice of a specific hypothesis class for use in the training process [87].

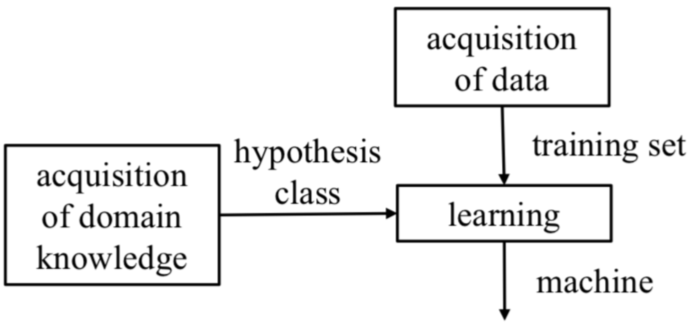

Figure 3.1: Machine learning methodology design flow [87].

3.1 Introduction to Machine Learning algorithms

Machine learning algorithms can be classified into four major classes depending on whether the task needs previous knowledge or the aim is to discover new patterns [89]:

- Supervised learning: These algorithms are used when labels of the data are known. The system receives as input a dataset with different feature values and the aim is to learn from this data in order to give an output. The correct values used for assigning a class to a new sample are the labels given by the user [90]. The algorithm infers a function which automatically maps the training set in order to classify new upcoming samples. In Supervised Learning, the task of classification is to find a rule which, based on human expertise and external observations, predicts the right label for any new input with its feature values. This type of learning is very useful in computational biology, where classification algorithms can predict mechanisms that are not sufficient well defined.

- Semi Supervised Learning: The aim of these algorithms is to predict the unknown labels from a dataset that is built up for classification purposes. A trained supervised algorithm is used to classify unlabeled data [91]. The most confident unlabeled samples and their predicted labels are added to the training set.

- Unsupervised Learning: It is used when the labels are unknown, as opposed to supervised learning algorithms. The training set consist of unlabelled samples, so that there no classes for splitting the feature space [87]. The aim of Unsupervised Learning is to observe the mechanics of the system and discover the insights, identifying populations of samples with similar features. Some examples of Unsupervised

Learning algorithms can be clustering methods, anomaly detection algorithms or neural networks on its unsupervised version [92].

- Reinforcement Learning: It is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems in a wide range of fields in both natural and social sciences, and engineering [93]. This type of learning is between supervised and unsupervised learning. Its goal is to choose the actions that maximize the future rewards [87]. According to Kälbling and Moore [94], "On each step of interaction the agent receives as input some indication of the current state of the environment. The agent then chooses an action to generate as output. The action changes the state of the environment, and the value of this state transition is communicated to the agent through a scalar reinforcement signal. The agent's behavior should choose actions that tend to increase the long-run sum of values of the reinforcement signal".

3.2 Clustering analysis

Clustering analysis classifies the samples from a dataset into subsets that have similarities among their own samples and dissimilarities from the rest of the subsets [95]. The samples in each subset, that can be represented by features in a multi-dimensional array, are placed according to some defined distances measured between each sample and the centroids of the clusters. Centroids are the centers of a cluster. Larger distances means less similarity to that subpopulation. Clustering algorithms are frequently used in data analytics, machine learning, pattern recognition and bioinformatics. Due to their immense applications, clustering methods can be classified into four main different categories [96]:

- Hierarchical methods: These methods create a hierarchical decomposition of the dataset and form a tree that splits the database recursively into smaller subsets[95]. There are two types of hierarchical methods: agglomerative, which start considering each sample as one cluster and iteratively merges pairs of them and divisive, which does the opposite [97].

- Density-based methods: Similar to the clustering methods that use the concept of distance in order to assign a sample to a cluster, but they use the density. This avoids spherical decisions on the feature space and finding clusters with random shapes. These methods are also useful to discard outliers or anomalies [98].

- Grid-based methods: These algorithms emerge as an improvement of density-based methods. The higher number of dimensions the dataset has, the less efficiency these methods provide. Using a grid-data structure the feature space is quantified into a finite number of cells, forming a grid structure where clustering algorithm operates [99].

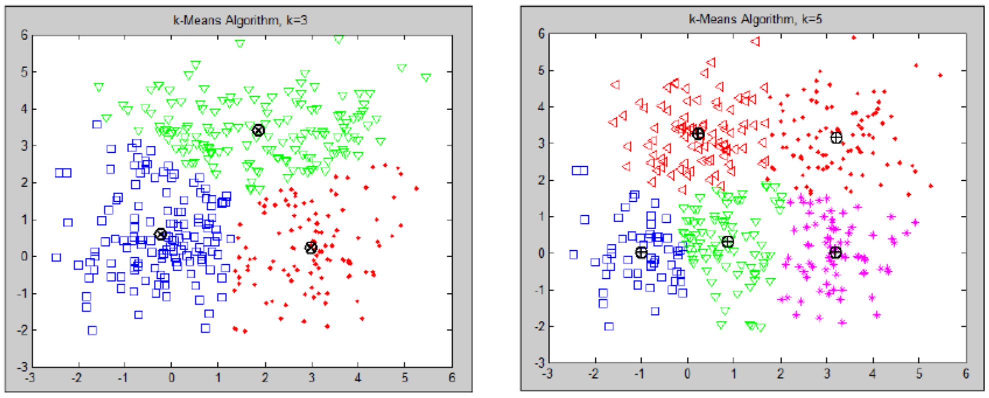

Figure 3.2: Clustering of samples in a feature space using k-means for k=3 (left) and k=5 (right) [101].

- Partitioning methods: Probably the most common used algorithms in clustering. These clustering methods divide the dataset into k number of clusters. Choosing a metric or distance function, like the sum of squared distances between all the dataset, the algorithm assign each sample to a unique cluster k [97]. That is the reason why clusters found with this type of methods are spherical. A typical example for such a method is the k-means clustering, which will be implemented in this work. The method k-Means is a NP-hard algorithm that partition the dataset into k clusters, where the mean observation in each cluster acts as the centroid [100]. The algorithm needs the k value as an input and iteratively give a cluster representation as an output. Fig 3.2 shows to runs using different values of k for the same dataset in its feature space.

After a clustering analysis it is important to evaluate the results. This is known as clustering validation. Its importance lies on the fact that clustering algorithms provide always a result, even when the dataset distribution does not form any clusters. That makes necessary to measure the efficiency of the method. Cluster validation can be internal (evaluating the stability of the clustering solution) or external (comparing the results with another datasets and methods). According to Tan et al. [102], there are some main steps in order to evaluate a clustering performance:

- Determining the clustering tendency of a dataset and distinguishing whether non-random structure actually exists in the data.
- Determining the correct number of clusters to the dataset.
- Evaluating how well the results of a cluster analysis fit the data without reference to external information.

- Comparing the results of a cluster analysis to externally known results, such as externally provided class labels
- Comparing two sets of clusters to determine which of the two is better.

3.3 Neural Networks

Artificial Neural Networks (ANN) belong to the Deep Learning (DL) scheme. DL a machine learning sub-field in which algorithms are inspired by the neurons in the human brain resembling it both structurally and functionally [87], [103]. Neural Networks (NN) are used to provide solutions to non-linear regression and classification problems, when the activation function is not linear. When a problem becomes more complex, the number of features increase exponentially and the linear combination between all of them needs to be taken into account. The computational cost is too high to deal with these problems with linear ML algorithms, but the development of more complex architectures can solve this problem. The first non-linear model was the Single Layer Perceptron, created by Rosenblatt [104]. This architecture consist of an unidirectional network formed by one input layer and one output layer. Selecting as activation function the sign function (or the sigmoid function) this algorithm is the most simple ANN created.

The feedforward neural networks, known under the name Multilayer Perceptrons, are a multilayered networks of perceptrons. This means, that several perceptrons are connected in series. In addition to the input and output layer, Multilayer Perceptrons have a hidden layer of neurons. The hidden and output layer consist of several perceptrons, which are called units. Fig 3.3 depicts an example for a feedforward neural network. With only one hidden layer with sufficient units and the right activation function, like the sigmoid function, the network can approximate arbitrarily close every continuous functions. Nevertheless, Deep Networks are empirically better and have a lower generalization error as shallow networks with one hidden layer. Due to the increasing computational power, it is possible to train a bigger network in shorter time. Thus is possible to test different structures and hyperparameters in a shorter time. Furthermore, the larger datasets assist a better generalization. Notwithstanding the great variety of NN algorithms that exist, we focus only on those that are most suitable for this project, Deep Neural Networks and Convolutional Neural Networks. Their convenience is explained in detail in the next chapter.

3.3.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are inspired by cognitive neuroscience and two researchers, Hubel and Wiesel, who work on the cat's visual cortex, which was found to have simple neurons that respond to small motifs in the visual field, and complex neurons that respond to larger ones [106]. CNNs are the most applied deep learning algorithms

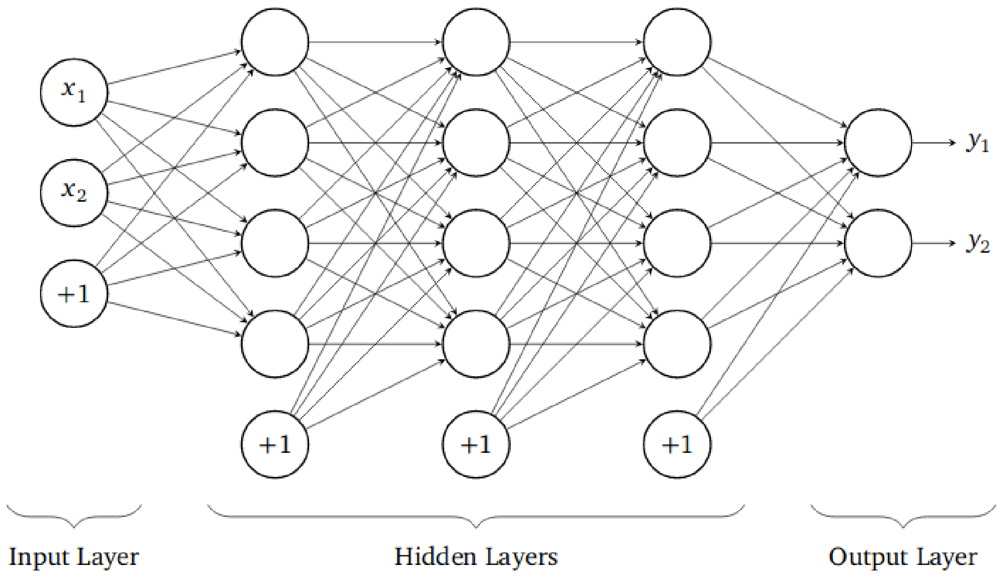

Figure 3.3: Example of a ANN architecture [105].

in Image Classification and are the core of most Computer Vision systems today. Natural Language Processing is another field where CNNs are applied with very promising results. In CNNs a convolution operation consist of the feature extraction from the input data done by filters that perceive descriptive conditions by scrolling trough the data and producing feature maps [87]. This extraction is done by the filters, a function applied to the data. The inputs of the CNNs are multi-dimensional arrays, such as two-dimensional images with three colour channels, or one-dimensional genomic sequences with one channel per nucleotide [107]. The high dimensionality of the images increase the number of hyperparameters to be tuned. Convolutional layers are also known as pooling layers, which allows the network to learn its own abstract features automatically. A typical architecture of a Convolutional Neural Network (DNN) is depicted in Fig 3.4. Hyperparameters such as the number of convolutional layers, number of filters or size of the filters must be tuned in the validation process.

3.4 Deep learning for genomics

Machine learning is a pillar of modern computational biology [107]. A large amount of effort has been put into sequencing the genome of living organisms. ML methods can be used to understand the genome, by identifying patterns in its sequence which are associated to regions with different functional roles. These patterns can provide for biologists new relationships between genomic sequence and phenotypes [109]. Nevertheless, the prediction of any region of the genome is still far from our understanding. Classical ML

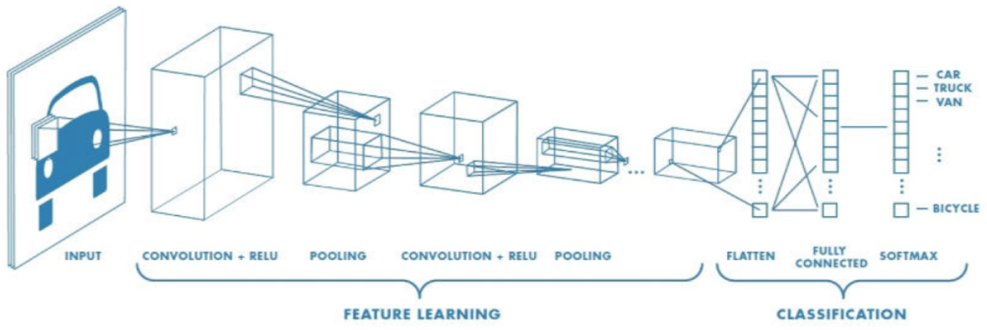

Figure 3.4: Example of a Convolutional Neural Network [108].

cannot operate as efficiently as Deep Learning methods due to the manual feature extraction previous to the training process. Deep Neural Networks (DNNs) can help circumventing the manual extraction of features by learning them from data [107]. Moreover, DL architectures can capture nonlinear dynamics from the translocation experiments.

Two of the most common useful NN configurations for learning from DNA sequence are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). RNNs have the power of dealing with time series problems, using the raw signal from translocations without feature preparation. On the other hand, CNNs can run directly over pictures like electron microscopy images, used for the study of DNA methylation [110]. CNNs allow to greatly reduce the number of hyperparameters compared to a fully connected NN by applying convolutional operations to only small regions of the input space and by sharing parameters between regions [107]. RNNs and CNNs can also be ensembled in order to create a network that combines the benefits of each type of network.

This project is not focused on the genome transcription, but on a deep scale which is basecalling. On the field of the basecalling, very promising methods DL architectures are emerging. From computational engineering, the first open-source basecaller based on neural networks was *DeepNano* [111]. This ML software uses bidirectional recurrent neural networks implemented in Python, using the Theano library [112]. The most famous real device for basecalling is the MinION by Oxford Nanopore Technologies. MinION is the first portable DNA sequencing device [83] that combines experimental translocations through protein pores with Deep Learning algorithms. This low-cost and portable device can achieve promising results in various applications including full human genome assembly [113] what could potentially lead to personalized genomic medicine. In MinION, nanopores are used to sequence DNA. An electrical potential is applied over a membrane that separates a bulk, in which the nanopore is inserted in order to move the DNA molecules from one chamber of the bulk to the other. As the DNA crosses through the pore, the sensor detects changes in ionic current caused by different nucleotides present in the DNA. The Oxford Nanopore Company currently have implemented on a cloud a platform called Metrichor, used to analyze generated sequencing data. In order to allow offline and private analysis of MinION data Nanocall open-source software can be used [114].

4 Methods

This chapter describes the experimental conditions, data collection and processing in DNA translocation experiments. Also the implemented methods for the clustering analysis and classification of DNA nucleotides.

4.1 Experimental Setup

In this work we have used experimental data from DNA translocation through nanopores. Specifically, two datasets are obtained from experiments of single DNA nucleotides translocations through 2D molybdenum disulphide nanopores [10], [11]. The experimental system consists of two chambers separated by a wall with a nanopore. Both chambers are filled with an ionic solution, cis chamber contains 100mM KCL and trans chamber a room temperature ionic liquid (RTIL). Fig 4.1 depicts the set up of the translocation experiments. The sizes of the MoS_2 pores are 2.8 nm for experiment B and 3.3 nm experiment A respectively. A voltage difference of 200mV is applied for both. The ionic concentration in the liquid is 0.1 M. To control the dynamics of the translocation, a viscosity gradient was applied to carry out the experiment by decrease the translocation speed. DNA datasets are provided by the Laboratory of Nanoscale Biology, Institute of Bioengineering, School of engineering, EPFL from Lausanne, Switzerland. Thanks to this collaboration, a machine learning approach has been developed in order to get new insights from physics of DNA translocation through 2D nanopores.

4.2 DNA data preprocessing

Two nanopore devices with different pore sizes have been tested under viscosity gradient conditions. Specifically, four different raw signals are obtained for each nanopore from the translocation of single nucleotides of the DNA, known as dAMP, dTMP, dGMP and dCMP (full form in Glossary pg. 51) [11], [115]. In order to interpret the ionic current signal and examine hidden physics in the traces it is necessary to identify every event concatenating all signals with an algorithm implemented in the Open Nanopore software, the CUSUM algorithm [39]. This method fits the raw signals from translocations experiments and provides structure files readable in numerical computing environments. In

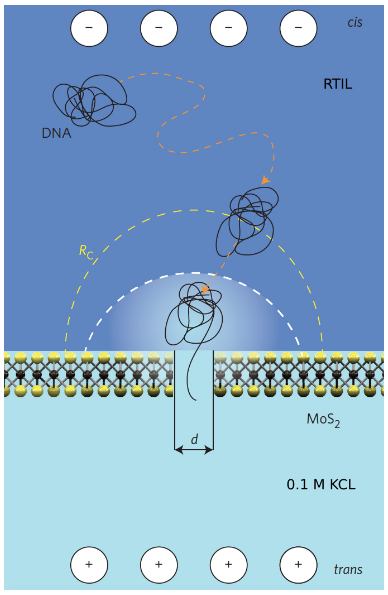

Figure 4.1: Viscosity gradient system formed by a MoS_2 membrane that separates the cis and trans chambers of the bulk. DNA motion away from the pore is diffusive. Within the capture radius R_c the nucleotides accelerate toward the nanopore due to eletrophoretic and electroosmotic effects. [11].

Fig.4.2 the unfitted concatenated signals for translocataion events of different nucleotides from experiment B with pore size 2.8 nm are plotted. Single events are marked using the information given by the CUSUM structure file. CUSUM algorithm provides a file which contains the concatenated raw signals, the same signal but fitted, time units, sampling frequency and the event data base. This latter contains the start and end point of each event, the number of events, number of levels from each event, the time of residence of the molecule in the pore known as dwell time and all the ionic current values during the translocation. Some random samples are included in the insets in Fig.4.2, showing the ionic current range values and the dwell time for each one. It is easy to distinguish the complexity of the event profiles compared to the theoretical predictions. Different types of events can be identified for each nucleotide dataset.

4.3 Feature selection

In order to analyze the different translocation events it is necessary to propose some candidates as potential features for the unsupervised learning model. Different combinations of those features will be plotted in the next chapter, creating feature spaces from the ionic current traces that will be analyzed. Physically intuitive aspects are considered to build the features, trying to include information of the dynamics of the events. In that respect,

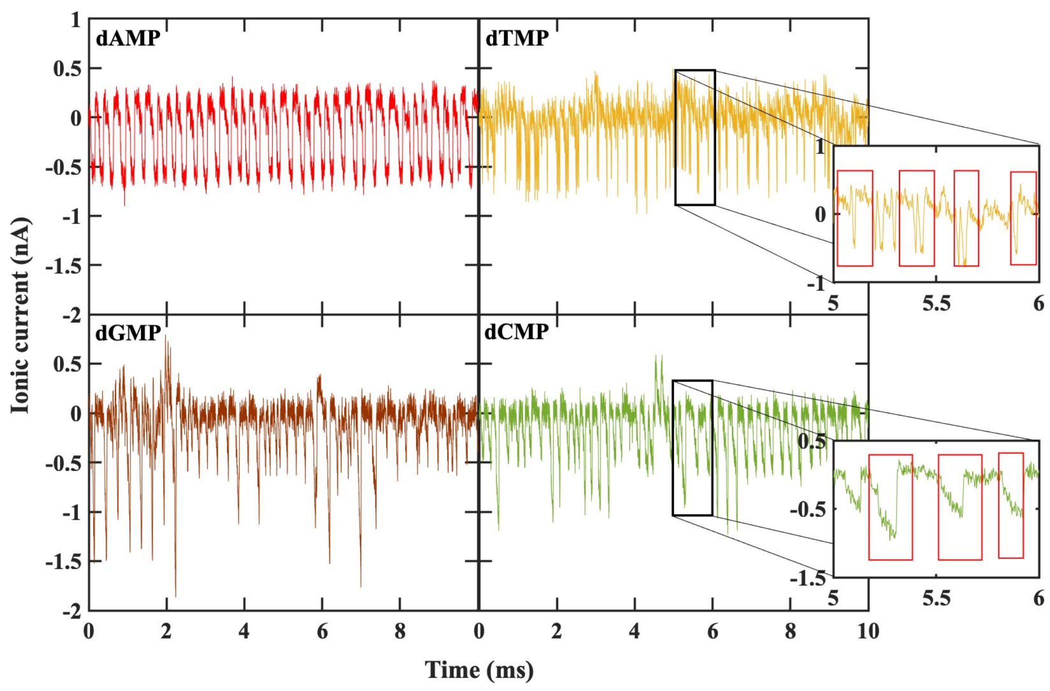

Figure 4.2: The raw ionic current signals with time for the DNA single nucleotides (a) dAMP, (b) dTMP, (c) dGMP, (d) dCMP ffrom experiment B. The insets show the concatenated events of single nucleotides translocating the pore. These single translocations are marked through the red rectangular regions [63].

for a possible clustering of translocation events, the following four features referring to single DNA translocation events are selected [63]:

(a) the traditionally used dwell or translocation time, which maps the duration of an event,

(b) the height of the ionic current blockade, defined as the difference between the maximum and minimum values of a single current blockade peak,

(c) the ionic blockade mean current (mean), defined through the average current value, and

(d) the levels, which are the number of the presumably different DNA configurations through the nanopore.

The levels contain information about the conformation of the molecule going through the pore, describing its topology. Different articles of nanopore experiments provide statistical analysis of the events using the dwell time and mean of the ionic current blockade [39], [44], [116]. The ionic current blockade height is the novel feature proposed to improve the efficiency of the clustering and classification algorithms. Its choice was physically intuitive in order to study new scenarios from the temporal series data. This new descriptor gives the possibility of creating, not only time dependent feature spaces, but also ionic current blockade dependent feature spaces which include information inherent in a single ionic current blockade. In the next chapters it will be shown that using the feature "height" it is possible to study the topology of single nucleotides during a translocation, comparing the number of clusters with the number of the most probable configurations obtained from the typical level analysis.

4.4 K-Means clustering

The main objective of this work is to propose an efficient feature in order to interpret the translocation events shown in Fig.4.2. The investigation is focused on searching pathways to better interpret nanopore data by clustering these using unsupervised learning techniques. In this way, one can identify patterns and similarities in the feature space [117]. The algorithm selected for this purpose is the open-source clustering algorithm k-means from the scikit-learn library [118]. k-means is a random-initialized iterative clustering method, which is widely used in data mining, vector quantization, data compression and pattern recognition [119], [120]. For a given k number of clusters, the algorithm can iteratively find centroid positions according to the data position in the feature space [120]. This project is not based on algorithm development, so k-means has been run without any modifications. Clustering methods such as k-means can be used for improving the error-ratio in a classification task, building a structure by clusters accelerating the training step. For DNA sequencing, the read accuracy is improved with k-means algorithm, reducing the systematic and random noise in the experimental datasets [121].

4.4.1 K-Means optimization

One of the inputs of k-means algorithm is the number of k clusters in the feature space. This can be optimized automatically by running enhanced versions such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm but, in order to have a better control of the data clustering, standard k-means will be used and not its enhanced versions that automatically select the best k number of clusters [120]. Instead, two different statistical scores are implemented to better tune the clustering process. The optimization of the number of clusters k is done using two well known clustering scores: the (a) silhouette (S) score [122] and (b) Calinski-Harabasz (CH) score [123]. The choice is based on their high intrinsic differences. In general, the S-score is a good statistical tool, but it fails with sub-clusters and very close or similar clusters [124]. On the other hand, the CH-score is one of the best schemes for identifying sub-clusters. Here, the value of k based on both the scores for all data sets is validated.

4.5 Neural Network Architectures

At another level, we use Neural Networks (NN) to optimize and accelerate DNA translocation. In this way, supervised deep learning model will take advantage of the feature extraction from the clustering method. Based on this analysis, different combination of features are used as inputs for the Neural Network models presented in the following subsections. Here, two different models are proposed: At first, a Deep Neural Network (DNN) for regression problems is implemented to accelerate the preprocessing step. Then, a Convolutional Neural Network (CNN) for classification is used to label new translocation events from nanopore reads. both methods will be done with the open-source deep learning library Keras [125]. This package works with TensorFlow [126] in the back-end to run the code automatically with the GPU and allow the application of many different types of NN architectures. In this thesis DNN and CNN will be used instead of Recurrent Neural Networks (RNN) or other configurations. Each translocation event has different dwell time value, being necessary to complete the arrays with white noise at the beginning and the end if we wanted to use RNN. Combination of features selected from clustering analysis makes possible to interpret each feature as a channel or pixel and build 1D images as inputs for the DNN and CNN. In the next subsections, the architectures for both NN models will be discussed, as well as the strategy for the optimization of the error-ratio. The DNN diagrams correspond to the first architecture selected. Based on this, we have further optimized the number of layers and other hyperparameters.

4.5.1 Data-to-Image transformation

In deep learning the use of NN makes sense if a problem can be imagined as an image classification. In this work, arrays with the values of the features are transformed on grey-scale level images, based on the idea in [108]. Santamaria et al. assign four different grey values in range 0-1 to the ionic current signal from different DNA nucleotides. As each DNA molecule contains the same number of nucleobases, it is possible to represent each sequence with a 2D image writing in each row the grey values and repeating n times that row, with n the number of nucleobases per molecule. In this project, instead of relating each pixel with an ionic current value for a discrete time step, pixels will correspond to the features selected in the preprocessing as channels. The data must be normalized between 0-255 in a grey scale. 1D images of 3-4 pixels will be used as input for the Neural Network models.

4.5.2 Deep Neural Network for height prediction

Deep Neural Networks can be applied in order to accelerate the feature extraction. In this case a regression algorithm is implemented, as the aim is to obtain numerical current values instead of categories. For this purpose, the Keras Regressor algorithm from the Keras Library is used. The configuration adopted is a DNN with 2 fully connected hidden layers of 5 neurons each. We use an input layer with 3 neurons which correspond to the features ionic current mean, number of levels and dwell time. The 1-neuron output layer is the predicted value of the ionic current blockade height. In Fig.4.3 the DNN is shown, where the individual nodes connecting the layers on the top are the bias terms of the cost function.

4.5.3 Convolutional Neural Network architecture

The aim of this part is the prediction of single nucleotides combining datasets from different experiments. In order to incorporate the sequencing information into the first layer for the input of the CNN, 1D arrays of 4 pixels corresponding to the features selected, including the height, have been extracted. The code structure implemented is based on the 2D Convolutional function to 1D from Keras[125]. Instead of using a 1D convolutional function, it is better to adapt the input and the size layers to a 1 dimensional structure in order to test in the future with 2D images when larger datasets are able. As this thesis uses small datasets, a 2D adapted convolutional function will be implemented to take advantage of the efficiency of Keras. The standard configuration before architecture optimization for the CNN is as follows: 1 input layer with 1 channel, 2 convolutional hidden layers with 30 filters, a fully connected or dense layer with different filter sizes and 1 output layer with 4 nodes that correspond to the single nucleotides dAMP, dGMP, dCMP and dTMP. For this last, is important to recode the original labels into 0-1 values, which is the

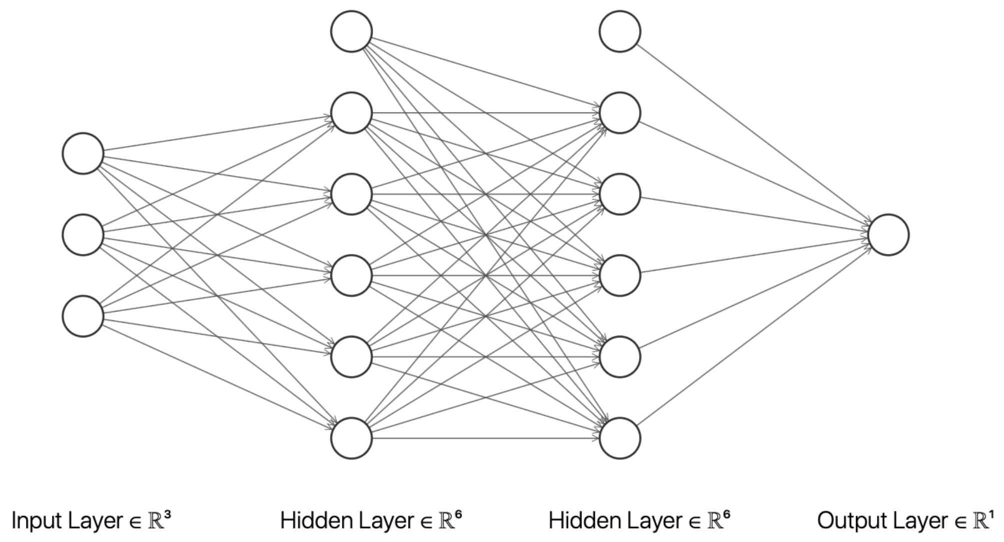

Figure 4.3: A DNN architecture for the prediction of the ionic current blockade height. The sketch shows the use of 3 features as inputs, 2 hidden layers with 5 nodes plus bias term and 1 output.

range of values that the activation delivers as a prediction. Fig.4.4 shows a sketch of the topology used for the CNN. Fig. 4.3, 4.4 depict the general topology of the NNs implemented in this work. The details of hyperparameters optimization will be discussed in the results chapter. Convolutional architectures in Keras need as input different parameters from the layers that must be tuned in the cross-validation step:

- The size of the first layer. It is interesting to start the CNN runs with the most common used feature in a problem and then add, change or delete other potential candidates for feature. This can be done manually or by feature learning.

- Number of filters in a layer. The number of filters is the equivalent to the number of neurons in a DNN, since each neuron performs a different convolution on each image.

- Flatten layer. It is the connection between the convolution and dense layers.This last is the layer used before the output layer.

- Dense layer. It is a fully-connected or linear layer type that is used in many cases for Neural Networks.

- Dropout layer. These layers are added between convolutional layers and work by dropping out probabilistically input nodes (or filters) to a layer, which saves computational time on the simulation.

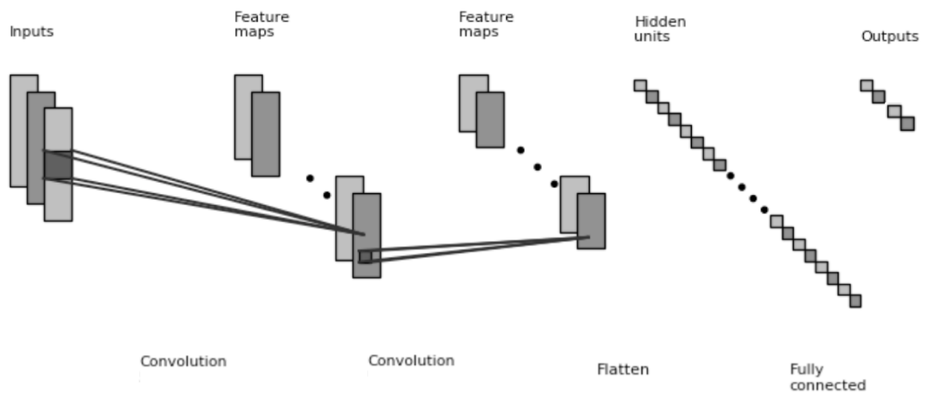

Figure 4.4: A CNN architecture proposed for the classification of single nucleotides. 2 convolutional layers, the input layer and 1 layer with 30 filters and 3x1 size, 1 flatten layer, 1 dense layer with 1024 filters and 1 output layer which delivers the recoded vector containing only 0 or 1 values, assigned to different nucleotides.

4.6 Neural Networks protocols

The training of a NN or CNN requires a number of hyperparameters to be tuned in order to study the possible bias or overfitting in the model. A hyperparameter tuning will be used here to decrease the error-ratio in the classification and regression. To this end, computing and plotting the training and cross validation error at every iteration is possible to check the loss and accuracy of the NN. In Table 4.1 a list of the tuned hyperparameters is presented. Keras present more hyperparameters easy-to-tune like batch size or pooling layers. The number of epochs define the number of times that the learning algorithm will work through the entire training dataset. Batch size is the number of samples that enter into the input layer each step. Larger batch sizes increase the accuracy although also the training time, as it requires more memory. using a batch size less than the size of the data set allows for the network to use less memory on every step and train faster. The pooling layer is a tool that downsample an image. A pooling layer is applied after one or multiple convolution layers. The convolution layers have the task to extract useful features from the input, which results in multiple feature maps. The pooling layer reduces the spatial

Hyperparameters	Range
Number of filters	2, 3, 4, 31, 128, 512, 1024
Filter size	2, 3
Number of layers	2-8
Number of Dropout layers	0-2
Number of units in the hidden layers	3-10
Number of epochs	100-2500

Table 4.1: A list of the hyperparameters to be tunned.

$$\text{Accuracy} = \frac{\#\text{TP} + \#\text{TN}}{\#\text{TP} + \#\text{FP} + \#\text{TN} + \#\text{FN}}$$

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}}$$

$$\text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}$$

$$F - \text{value} = \frac{2\text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Figure 4.5: Statistical scores to study the accuracy in a NN [127].

size of these feature maps [105]. The way to measure the performance of our algorithm is by using statistical scores. Keras has already implemented different Loss functions in order to see the overfitting of the training and test. As part of the optimization algorithm, the error-ratio must be calculated repeatedly. This requires the choice of a loss function, that can be used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. In order to calculate the accuracy, Fig 4.5 shows the classical statistics in deep learning.

5 Clustering Analysis

In this chapter, the clustering results will be discussed. Insights into different molecular events and topologies thorugh the nanopore will be gained.

The next section provides an explanation about the selected feature space. The efficiency of the novel feature ionic current blockade height is explained plotting two dimensional feature spaces.

5.1 Feature extraction

Our first aim is to find the proper feature space in order to better interpret the experimental data. Scatter plots for single nucleotide translocations are used in order to identify their different configurations. This will help to select the most relevant descriptors as inputs for the classification model. The number of events for those datasets from experiment B are 3887, 127, 119 and 672 for dAMP, dTMP, dGMP and dCMP respectively. From experiment A 22, 240, 757 and 391 events for dAMP, dTMP, dGMP and dCMP are obtained. Different combinations of the dwell time, mean, and height have been examined. The scatter plot in Fig.5.1 corresponds to the combination of the ionic current blockade height, ionic current mean and dwell time for dAMP from experiment B. Note that typically the dwell time of the translocation events is used [39], [45]. Overall, choice of features is crucial. In clustering methods the standardization of features can be problematic, as it can bias and modify the data changing the distances between the samples leading to different clustering combinations. In order to avoid this, no scaling techniques will be applied. On the other hand, for supervised learning, the scaling of features is often necessary in order to accelerate the learning process [128]. We analyze first the feature space by taking the combination of the mean and dwell time. The results are shown in Fig.5.1(a) and do not point to clearly distinct clusters that could lead to possible configurations of dAMP through the nanopore. The combination of the mean current with the dwell time only identifies a large cluster, while the combination of the dwell time with the height generate two clusters, which are overlapping. This configuration is known as subclusters. Fig.5.1(b) shows what happens when dwell time is not used. Two clear clusters map two different types of molecular configurations through the nanopore. Fig.5.2 shows the

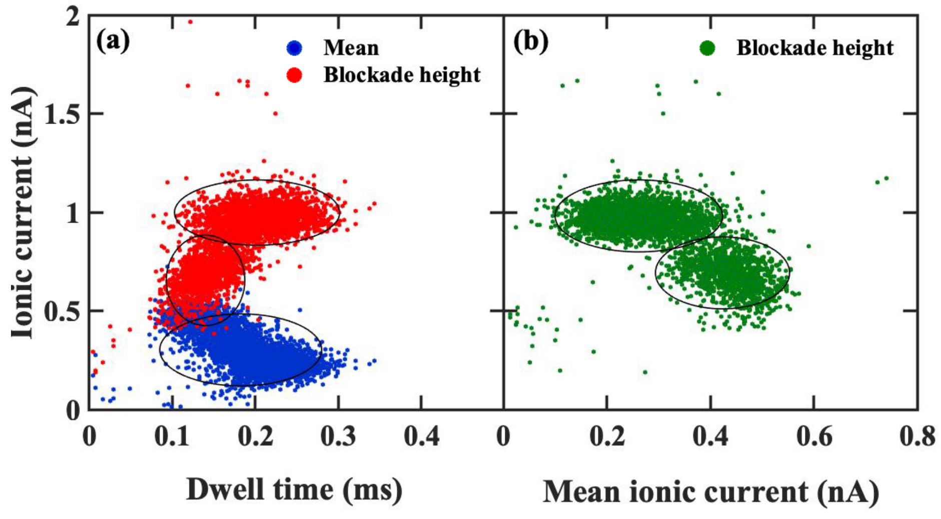

Figure 5.1: Feature space analysis using the data from the dAMP experiment B in Fig.4.2. Different combinations of the dwell time, mean ionic current, and blockade with height are taken as denoted by the legends. The black ellipses define possible clusters [63].

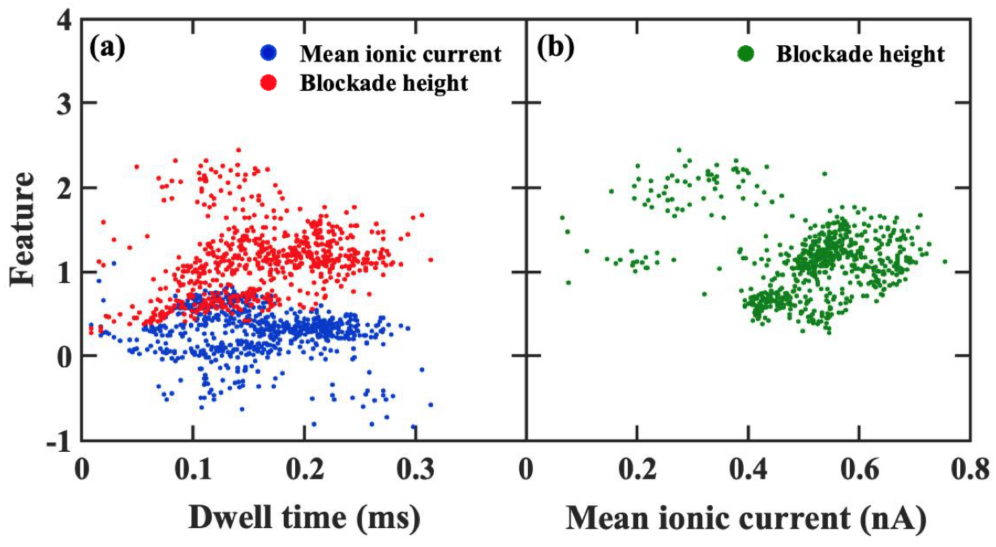

Figure 5.2: Feature space analysis using the data from the dGMP from experiment B in Fig.4.2. Different combinations of the dwell time, mean ionic current, and blockade with height are taken as denoted by the legends.

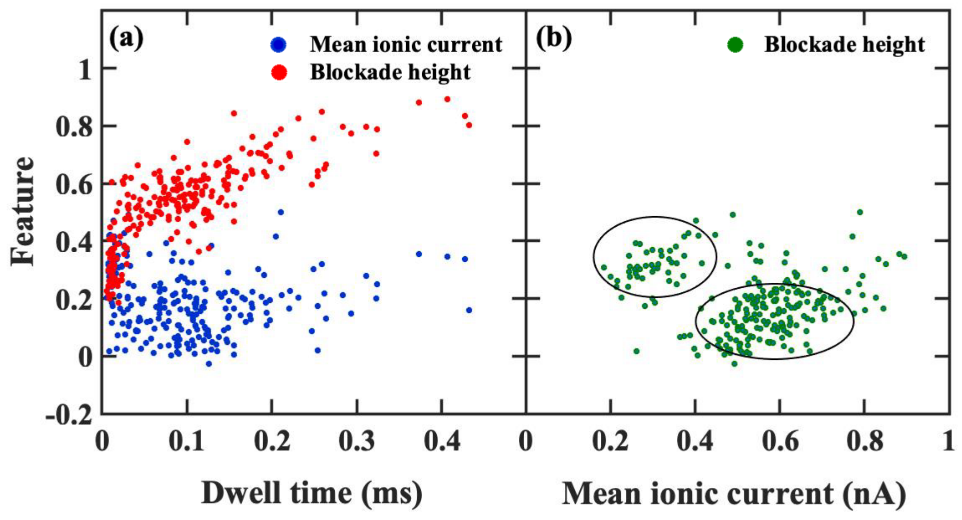

Figure 5.3: Feature space analysis using the data from the dTMP from experiment A. Different combinations of the dwell time, mean ionic current, and blockade with height are taken as denoted by the legends. The black ellipses define possible clusters.

same behaviour of the feature height and its different combinations for the nucleotide dGMP. In this case, the new feature "height" separates better the clusters. Scatter dTMP data from the experiment A has been also plotted in Fig.5.3, in order to demonstrate the scalability of the method using the height, which identifies clusters not depending on nanopore materials or sizes. Accordingly, the blockade height feature has the quality of clearly being clustered providing better information than the commonly used dwell time. This emphasizes the novel feature of the approach proposed here.

5.2 Clustering visualization and feature efficiency

Based on the feature extraction and analysis mentioned above, the k-means clustering algorithm is applied to the translocation data from all nucleotides of experiment B. The feature space analysis with respect to blockade mean and height for all the single nucleotides is given in Fig.5.4. Indeed, two distinct clusters can be seen in the case of adenine demonstrating two distinct molecular configurations for dAMP. For the other nucleotides, the picture is similar, though not always as clear as for dAMP due to the higher sparsity of the other data. In any case, even for the few data for dGMP two distinct clusters seem to form, while dCMP has the tendency of forming four clusters.

In order to find the optimum number of clusters for all data sets and confirm the findings above, the Silhouette (S) and Calinski-Harabasz (CH) scores are calculated and cluster sizes k in the range 2-10 are scanned. The results for all data sets are summarized in Fig.5.5. The scores in this figure are obtained by taking two sets of features, the blockade height vs. dwell time and the blockade height vs. mean. For visibility and comparison with the S-score, the CH-score was scaled using the Max-Min Normalization technique within the [0-1] range [129]. For a given data set, the highest values of these scores represent the most probable number of clusters. Similar S-score values for different k values point to the existence of sub-cluster configurations. In order to further evaluate this, it is necessary to turn to the CH-score in the same k range. Note, that the aim is not to focus on the best possible way for optimizing classifiers, rather on finding the global density clusters of the translocation data. As evident from this figure, the choice of the dwell time in the feature space leads to an almost stable S-score value as increasing k for all nucleotides. As opposed to this, a linear increase is observed in the CH-score. The different behavior of the two scores underlines the fact that the dwell time is not a good feature for classification. In contrast to this, when choosing the blockade height and the mean current as the feature space, both scores show a similar variation with the number of cluster size. For dAMP, the S-score is similar for cluster sizes $k = 2$ and $k = 3$. For the CH-score a cluster size of $k = 2$ shows the largest score. This k is related to a higher score also for dTMP and dGMP. Accordingly, dAMP, dTMP, and dGMP assume two distinct types of configurations through the pore as denoted by the cluster size of $k = 2$. A deviation can be observed for dCMP, for which the CH-score is higher for larger cluster sizes. In order to visualize this in Fig.5.4 cluster sizes of $k = 3$ and $k = 4$ are sketched for dCMP. These denote the high probability of finding four different types of dCMP configurations in the pore. Overall, when the number of clusters increases, the CH-score is higher than the S-score due to the small distance of the samples to the cluster. Applying the k-means algorithm in sparse data sets cannot distinguish among optimum cluster sizes when using the feature space of dwell-time and mean current. Nevertheless, including 'blockade height' in the feature space (especially without the dwell time) leads to a more efficient classification even for the more sparse data sets. In this case, the variation of both scores with k is similar supporting the higher efficiency of 'height' over 'dwell time'.

5.3 Interpretation of the clustering results

In order to benchmark the cluster sizes in Fig.5.5, it is necessary to compare within the information obtained through an analysis of the current blockade levels and events. In this way, a physical meaning can be assigned to the clustering obtained through our ML

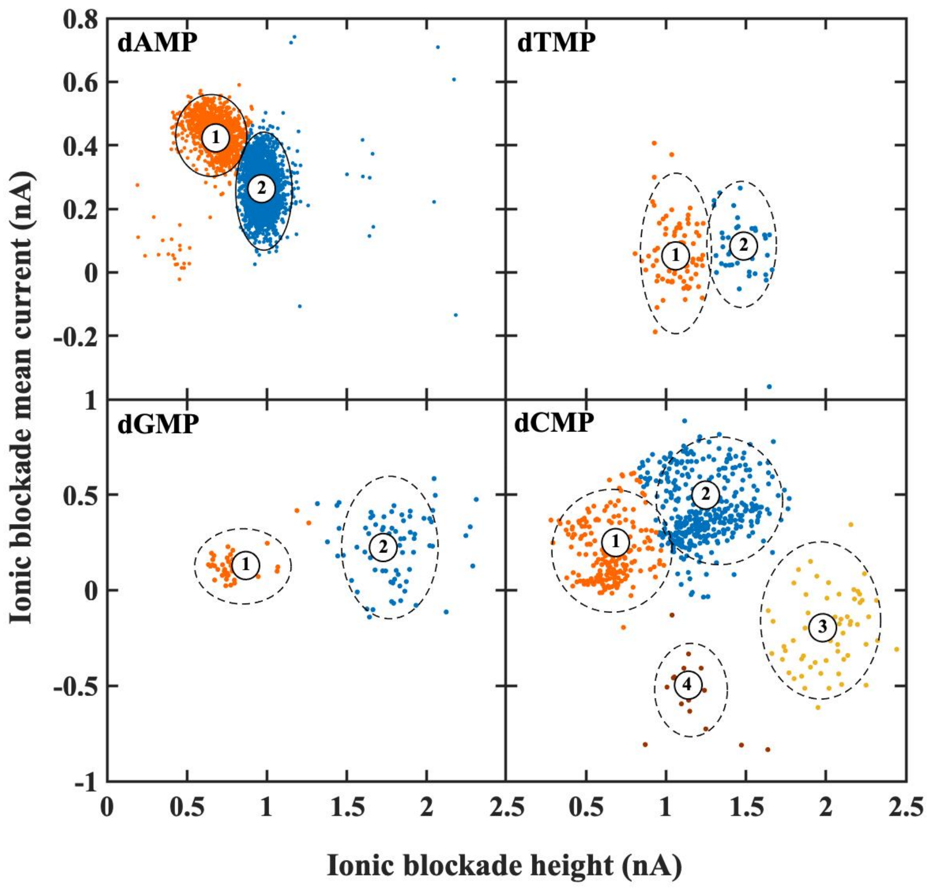

Figure 5.4: Feature space analysis for the (a) dAMP, (b) dTMP, (c) dGMP, and (d) dCMP translocation experiments from Fig.4.2 [63].

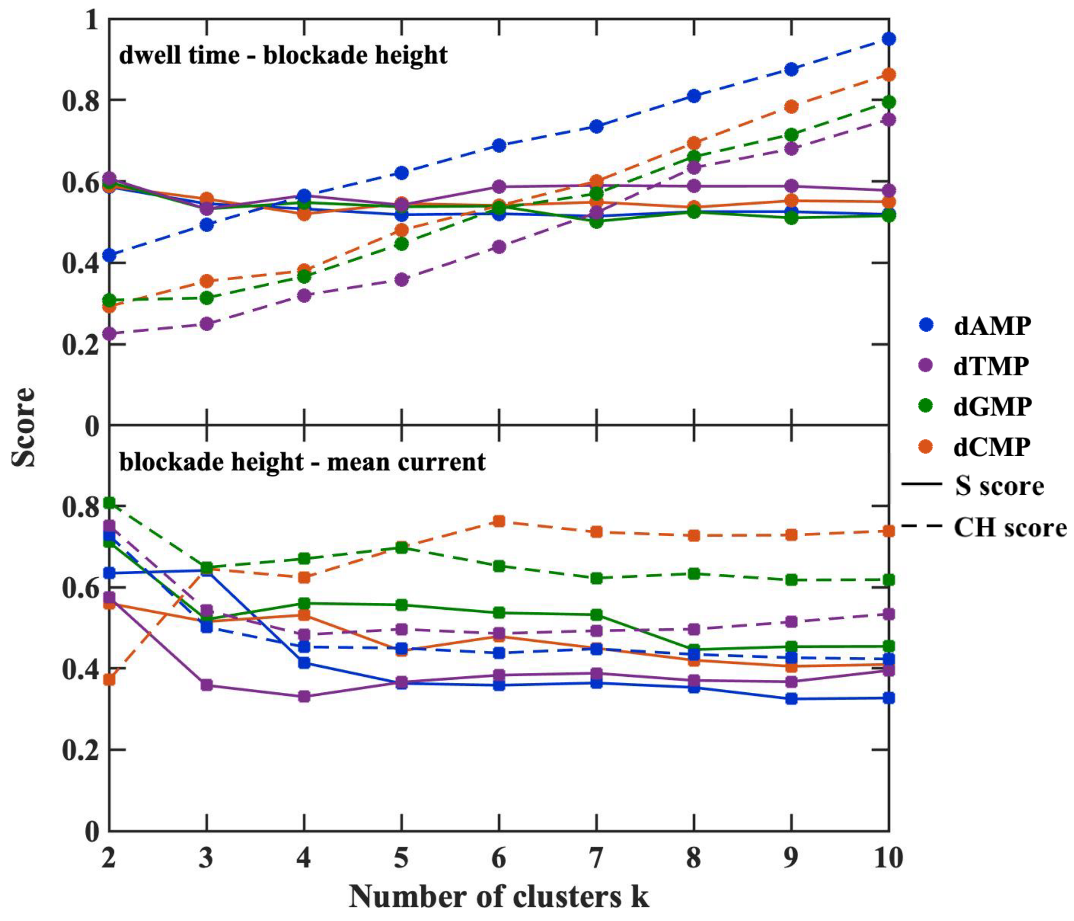

Figure 5.5: The Silhouette (S) (solid lines) and normalized Calinski-Harabasz (CH) (dashed lines) scores for all single-nucleotide translocation experiments. The scores are obtained from the analysis of the feature spaces dwell time - blockade height (filled circles) and blockade height - mean current (filled squares), as denoted in the panels [63].

approach. For this, the current blockade levels is considered instead of the mean current blockade, attempting to classify the number of configurations observed.

For the level analysis, OpenNanopore software [39] has been used for extracting statistics from the translocating data. In this way, the length of the data is increased by resolving also the number of different current blockade levels within a single current blockade. A scatter plot from this level analysis as a function of the dwell time and the ionic current data for dAMP from experiment B is shown in Fig.5.6. The data is represented through a histogram revealing two distinct peaks. These two peaks arising from the levels correspond to two most probable conformations of a dAMP threading the nanopore. These results again support the fact that using the current blockade height values of the translocation levels instead of the events themselves leads to similar results and is a very efficient approach for extracting information on the molecular conformations in the pore. The same correlation is found in Fig.5.7, corresponding to dTMP data from experiment A. The scatter plot of this data in Fig.5.3 shows again the correspondence between the peaks from the level analysis and the clustering method. Can be concluded that this method not depends on the pore size.

The representation in the levels figures clearly denotes that the most dense cluster in a distribution maps a higher occurrence probability of a certain molecular conformation. In the case to longer DNA molecules, such two peaks would be related to unfolding (lower current, slower translocation) and folding (higher current, faster translocation) conformations [26], [130]. In the case of single nucleotides, these peaks can be assigned to either different entrance configurations of the molecules in the pore or possible molecular isoforms. This remains to be shown in a separate study. Here, the aim was to provide a feature analysis without the need of a levels analysis. It is possible to reveal a direct robust relation between the molecular topology and the feature height.

5.4 Dealing with sparse datasets

Another very important aspect is the handling of sparse data sets. Intuitively, the larger the data set the better the clustering of the events. The significantly higher efficiency of the blockade height in clustering nanopore data has been shown here (see Fig.5.1). These results, though, were shown for the rich dAMP data sets. It is important to support this findings by performing the same analysis for the much more sparse data for dGMP from experiment B in Fig.5.8. Overall, the use of the blockade height can identify two clusters. These are very distinct when the second feature is not the dwell time. The exclusion of the height in the feature analysis cannot identify more than one clusters, despite the fact the better Silhouette score resulted in an optimum size of two clusters (Fig.5.5).

As a final note, the fact that the height feature is more efficient than the traditional dwell time for clustering events relies on the implicitly more statistics included in this feature. Specifically, to each ionic blockade the difference of the start and end stage of the

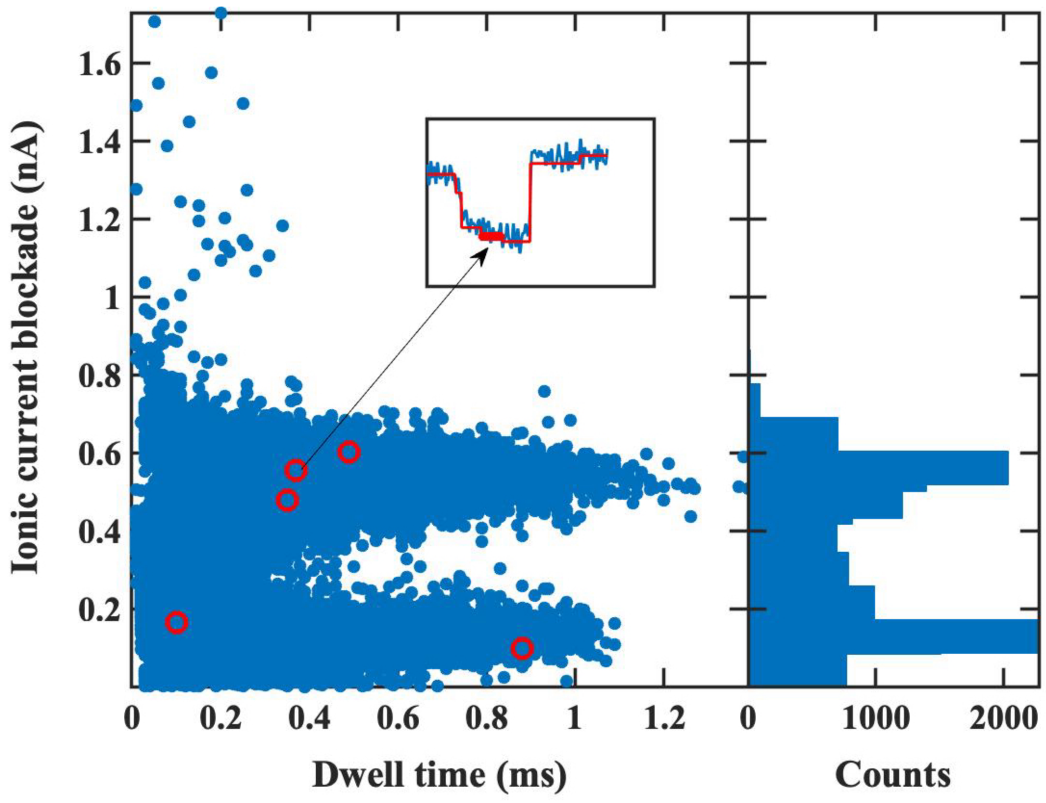

Figure 5.6: Blockade level analysis for dAMP showing all experimental data (left) and their distributions (right). Two distinct peaks representing two most probable configurations can clearly be identified. In the inset, the red lines correspond to the separate levels taken for one event. These levels point to the respective circles in the events plot [63].

translocation is assigned when taking the dwell time or the mean blockade. However, the blockade height is defined through the difference between the highest and lowest peak within a blockade. In this way more distinct information is included for each event. As clearly seen from the insets of Fig.4.2, there is a much more rich dynamics in the many levels (i.e. peak heights) in a single blockade and these are very different for each event. These multi-level characteristics typically defined by the blockade extrema are explicitly taken into account in the height feature. In terms of the underlying Physics of the translocation process, one could argue that from the three features examined here, the dwell time and mean blockade are essentially coarse-graining the detailed features of each blockade and through this the molecular details in the pore. On the contrary, the blockade height is inherently including the characteristics of the detailed Physics of a single translocation event. The extrema in the single blockade taken for the height definition are the ones characteristic of the process. These carry for example, information on the two extreme configurational changes of the molecule through the pore (relevant to the peaks in Fig.5.6). In

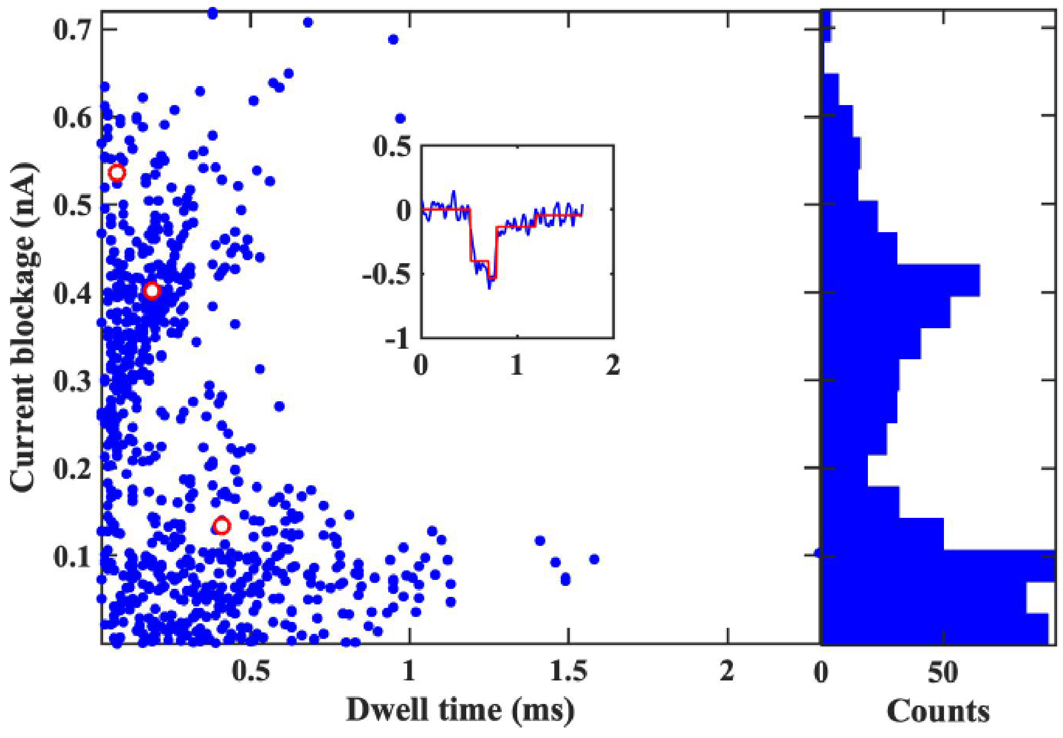

Figure 5.7: Blockade level analysis for dTMP from experiment A with pore size 3.3 nm showing all experimental data (left) and their distributions (right). Two distinct peaks representing two most probable configurations can clearly be identified.

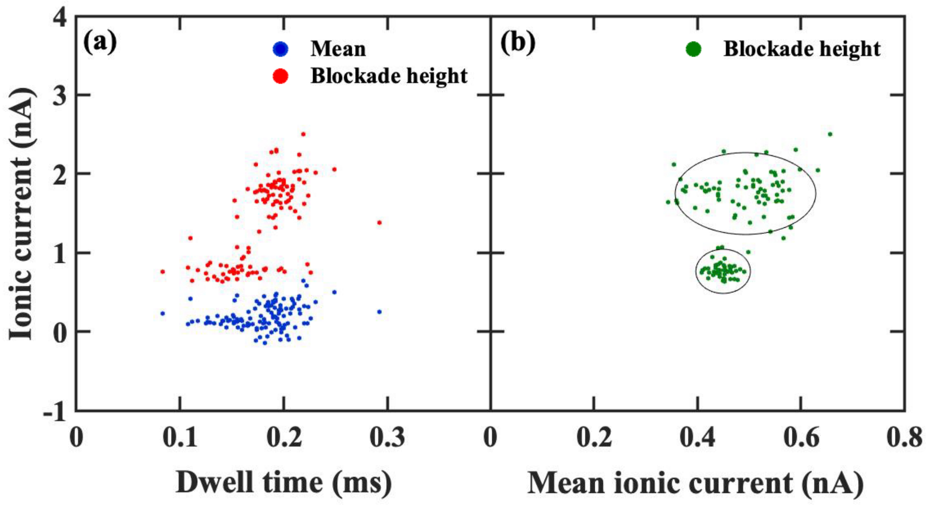

Figure 5.8: Feature space analysis using the sparse data from the dGMP experiment B in Fig.4.2. Different combinations of the dwell time, mean ionic current, and blockade with height are taken as denoted by the legends [63].

this respect, the blockade height includes much more rich dynamics than the dwell time and enhances the differences among different molecules.

6 Training Neural Networks with nanopore data

This part provides the results obtained from different neural networks architectures. The prediction of the novel feature height is shown, while the second part identifies between nucleotides selected from different experiments.

This chapter assembles the clustering analysis with the NN results. In the first part, the optimization of the DNN for predicting the feature height is shown. This can accelerate the preprocessing of the deep learning model. The second part plots the loss and accuracy value from different hyperparameter ranges, being possible to choose the best topology of the CNN for nucleotide identification.

6.1 Feature prediction

Deep learning models applied to small datasets are more difficult to optimize, as they do not behave like larger datasets while applying the common protocols for optimization in order to avoid the overfitting. In this section a DNN for regression will be implemented, using the largest dataset available from experiment B formed by 3886 samples, the single nucleotide dAMP. In the Table 6.1 the details of the dataset for training the neural network is shown. The aim of this part is to predict feature values and compare the efficiency of each. Concretely, the ionic current blockade height versus the classical features used in the statistical analysis of nanopore data. This method can be also used to accelerate the preprocessing, which makes sense if the dataset size can be increased with more translocation experiments. Rather, we show here a protocol to corroborate the conclusions extracted from chapter 5 about the ionic current blockade height.

Nucleotide	Experiment	Training	Test
dAMP	B	3000	887

Table 6.1: Number of samples from dAMP dataset for training and test.

The model consist of a simple DNN and the architecture has been optimized as shown in the first column of Table 6.2 along with the accuracy values. The activation function selected is the well-known ReLu, while the algorithm for minimizing the cost function is Adam [131]. The loss is measured with the mean squared error. Scikit-learn library

Classifier	Accuracy
Random Forest	0.6703
NN 5 nodes 1 Hidden Layer (HL)	0.8129
DNN 5 nodes 2 HL	0.8640
DNN 5 nodes 3 HL	0.8710
DNN 10 nodes 5 HL	0.8788
DNN 10 nodes 10 HL	0.8677

Table 6.2: Accuracy values for different classifiers.

provides useful machine learning algorithms for regression of fast implementation, as Random Forest (RF) [118]. This method is used for comparison with different NN architectures. It is clear the more graph complexity of NNs compared to RF improves the accuracy. Within NN, there is no hardly difference on adding more layers or more nodes. Also the computational time increase is not really relevant due to the dataset size, so an intermediate configuration has been selected for predicting feature values.

Two different combinations of features as inputs have been tested, choosing 3 nodes for the input layer. One set of features including the blockade height in order to predict the dwell time and other with the classical feature (ionic current mean, dwell time and number of levels) for blockade height prediction. For the training, the number of epochs used is 200, while the batch size is 0 as it is not relevant due to the small dataset and the computational time will be almost the same. In the second column of Table 6.2 the test validation accuracy only for blockade height prediction is shown for different NN architectures and RF for comparison. Here we check the problem of dealing with small datasets. The difference of accuracy between a common machine learning method like RF and a deep learning algorithms is clear, as DNN are created for more complex non-linear problems. Adding more layers improve the accuracy only until 3 hidden layers. Beyond those, the accuracy value stays at the same value. This gives us hints about possible bad quality of the descriptors, because the model does not improve while tuning its hyperparameters. The second test, using the blockade height along with the number of levels and the ionic current mean, predict better values as blockade height capture more rich dynamics from the experiment. This can be seen in Table 6.3.

Classifier	Output	Accuracy
DNN 5 nodes 3 HL	Height	0.8710
DNN 5 nodes 3 HL	Dwell time	0.9885

Table 6.3: Comparison between both tests for feature prediction.

This section ends with a typical plot in regression problems shown in Fig 6.1. It is interesting to see that the separated areas corresponding to the number of clusters from k-means analysis can be clearly seen in Fig 6.1 a). On the right, the results support the score analysis from chapter 5. Ionic current blockade height increases also the efficiency of the regression model.

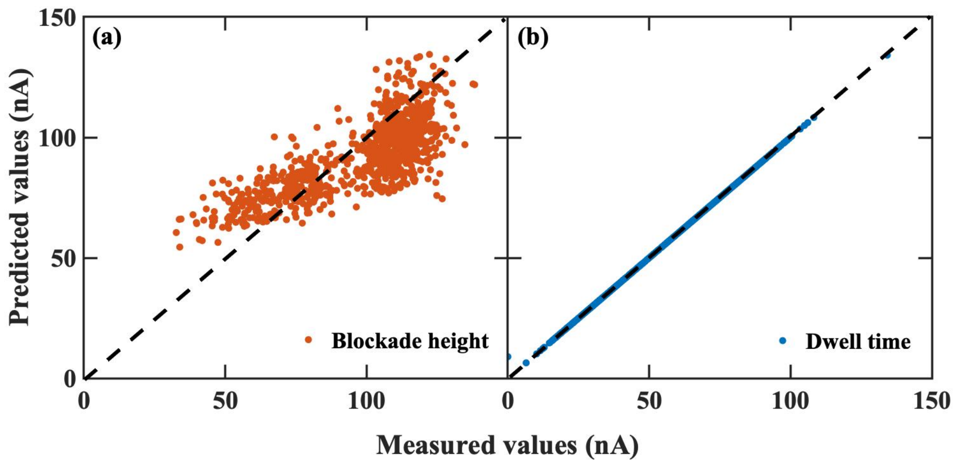

Figure 6.1: Predicted vs. measured feature values visualizing the accuracy of the DNN. Subplot(a) depicts the blockade height predicted values versus measures values with 0.8710 accuracy. On the left, subplot (b) can perfectly predict the dwell time using our novel feature blockade height, with a 0.9885 of accuracy.

6.2 Nucleotide prediction

In this subsection, the aim is to optimize the structure of a 1D-CNN in order to classify nucleotides from different experiments. The input of the 1D-CNN are 1D images, arrays of one dimension where each pixel is one channel or feature. The features are previously standardized in order to transform small current values into grey scale values. Usually it is necessary to fill arrays with random noise in order to run a neural network such as RNNs. This introduce artificial values into the dataset, losing the purity of the experiment. This method inspired by [108] allows to translate nanopore information into abstract grey level images without using time dependence and using only the translocation data. Very small 1D images will introduce as inputs in our 1D-CNN. We want to combine in the same training translocations from experiments with different conditions, as the pore size. Pore sizes are 2.8 nm for experiment A and 3.3 for experiment B and the ionic raw signals are absolutely different, there is no apparently correlation between signals from the same nucleotide going through both pores. Table 6.4 shows all the data available for simulations. In order to merge both experiment on one training set, we try to select number of samples per nucleotide and also the number of output nodes as larger as possible. It is clear the best combination is to set 500 samples per nucleotide and use as nucleotides Adenine and Cytosine from experiment B plus Guanine from experiment A. Below that, could be interesting to choose also Cytosine from A and 300 samples per nucleotide, although the results would not be reasonably robust. In Table 6.5 the training and dataset is shown. 1500 images form the training set and 450 the labels. The number of input units for the CNN will be 4, corresponding to the dwell time, the ionic current mean, the number of levels and the ionic current blockade height from each nucleotide. The number of output nodes will be 3, according to the three different merged datasets from experiments. The activation function is the same as for the DNN, again using the optimization algorithm Adam [131]. In order to measure the loss value the categorical loss entropy (CCE) has been calculated [132].

Nucleotide	Samples Exp. A	Samples Exp. B
dAMP	22	3887
dCMP	391	672
dGMP	757	119
dTMP	240	127

Table 6.4: Dataset sizes from experiments A and B.

Nucleotide	Experiment	Training	Test
dAMP	B	500	150
dCMP	B	500	150
dGMP	A	500	150

Table 6.5: Number of samples from each nucleotide dataset for training and test.

6.2.1 CNN Hyperparameter tuning

In order to optimize the topology of the CNN and deal with the overfitting, we have tuned the hyperparameters named in chapter 4. In Table 6.6 the workflow is shown. First, we have set the number of filters per layer. The more filters the CNN use, the larger is the simulation. Every filter used has a size of 3. Trying with 4 filters the algorithm does not converge every time, is not stable and needs around 2500 iteration when it converges. There is no problem on increasing until the CNN becomes stable due to the number of samples used. Using 1024 filters the algorithm converges in every run in a reasonable time for 300 epochs. Furthermore, the concept of filter in small 1D images is more abstract than in a usual 2D image. The problem is to set a value that allows to run on a robust way. The next step is to optimize the number of hidden layers. In the same way as the DNN, we do not observe differences adding more layers, so the number of hidden layers is set on 2. The number of epochs used is the necessary until convergence for all cases. It is important also to shuffle the input data in order to break the symmetry of the initial weight values. We do not have tuned the batch size because the computational time is brief with size zero. That would not be possible while working with 2D larger images.

HL	Filters	Dropout	Reg. Param.	Epochs	Loss	Accuracy
8	4	Yes	0	2500	2.1310	0.8733(!)
8	250	Yes	0	300	2.0427	0.8622
8	512	Yes	0	300	1.9274	0.8688
8	1024	Yes	0	300	2.0746	0.8660
7	1024	Yes	0	300	2.0919	0.8731
6	1024	Yes	0	300	2.037	0.8666
5	1024	Yes	0	300	1.9588	0.8622
4	1024	Yes	0	300	1.9588	0.8666
3	1024	Yes	0	300	2.1076	0.8600
2	1024	Yes	0	300	2.0890	0.8578
2	1024	No	0	100	2.0935	0.8577
2	1024	No	0.00001	100	1.9871	0.8689
2	1024	No	0.0001	100	1.7719	0.8711
2	1024	No	0.005	100	0.8398	0.6355(!)
2	1024	No	0.001	100	0.6247	0.8688
2	1024	No	0.01	100	0.6795	0.8555(!)
2	1024	No	0.1	100	0.59803	0.7088
2	1024	No	1	100	1.1333	0.3333

Table 6.6: Accuracy results for different architectures of 1D-CNNs.

CNNs usually introduce dropout layers in order to unload the network of the nodes with the smallest weights [133], decreasing the simulation time. This case surprisingly is the opposite, due to the small size of the 1D images. In Fig 6.2 this effect of non using a dropout is depicted. The algorithm needs less epochs until convergence without dropout. Deleting the dropout layer we only use 100 epochs instead of 300 we used with it. After selecting the best configuration for the 1D-CNN is mandatory to decrease the Loss value in order to avoid the overfitting. The regularization parameter has been tuned, as shown in Table 6.6. Exclamation marks denotes unstable simulations. The optimal value for the regularization parameter is 0.001. There is some intrinsic overfitting on the model that can be seen in Fig 6.3, as the 1D-CNN cannot improve its performance more than 0.88 in all the cases.

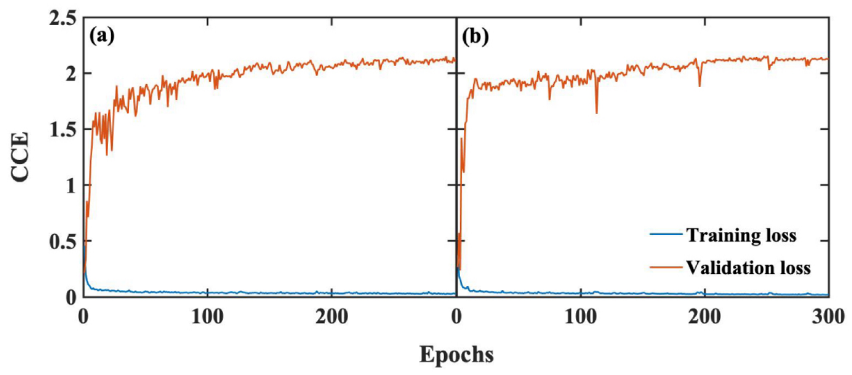

Figure 6.2: Accuracy values with (a) and without (b) dropout for a CNN with a dense layer built by 1024 filters.

In view of these results it is promising with only 1500 samples, although the model still overfits. However, the loss convergence can be improved. This improvement would be better achieved with more rich datasets than by increasing the number and type of features.

6.2.2 Importance of data selection in small datasets

Outliers in deep learning with small datasets can be a problem. While working with larger datasets is not relevant to select the best quality samples from your dataset. Nevertheless, small datasets can contain a range that makes the model overfit, as this case. We found, after testing different sample ranges within the nucleotide sets, an anomaly in the Cytosine dataset. The first 100-150 measurements avoid for the CNN to learn properly. Using the last 500 samples instead of the first 500 the accuracy increase until 0.9885 while the loss value decrease even more. Fig 6.4 depicts the performance of the CNN with different Cytosine training samples and Table 6.7 compares the accuracy from those datasets, proving the importance of sample selection with small datasets.

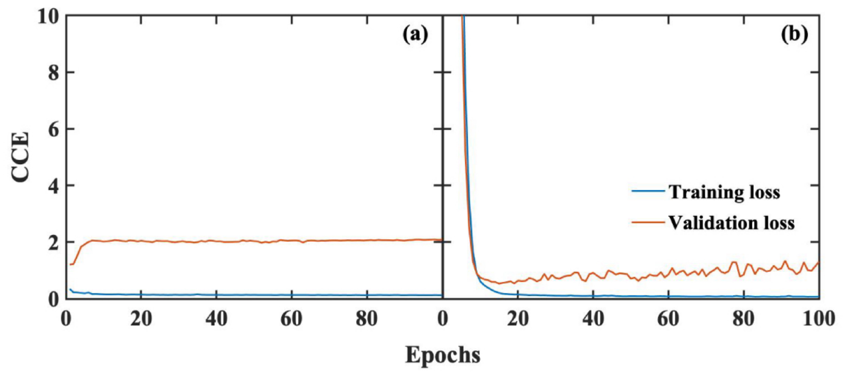

Figure 6.3: Loss values without (a) and with (b) regularization for a CNN with a dense layer built by 1024 filters.

Classifier	dCMP samples range	Accuracy
1D-CNN 2 HL	0-500	0.8710
1D-CNN 2 HL	150-650	0.9885

Table 6.7: Comparison between tests for nucleotide prediction using different Cytosine samples.

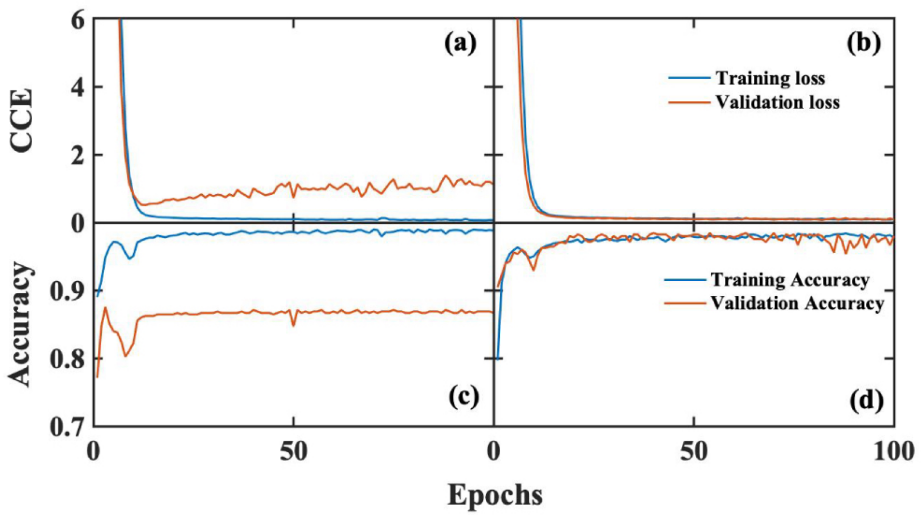

Figure 6.4: Categorical loss entropy and accuracy values of 1D-CNN simulations for different Cytosine training samples. a) and c) subplots depict CCE and accuracy from the simulation with Cytosine samples in range 0-500, which induce overfitting. In b) and d) subplots, the improvement after selecting the most relevant samples for training is depicted, where loss decrease while the accuracy increase.

7 Conclusions

This chapter shows the conclusions from the machine learning model and future lines.

In this work, a new feature used with an unsupervised and a deep learning model for identifying different molecular topology from translocation events of DNA through ultrathin MoS_2 nanopores has been proposed. The novelty of this approach is based on using the ionic blockade height instead of the 'traditional' dwell time or the level information. It is shown, that the commonly used dwell time is not the best feature for unraveling molecular events in the pore. Instead we could manifest that the novel feature of the ionic blockade height is highly more efficient in forming well defined clusters of events. In order to support this, a classification for clustering experimental data for different DNA molecules is performed. The approach does not use labeling of the data, thus is not biasing the leaning process. It is possible to distinguish types of most probable molecular conformations of the DNA threading the pore. Accordingly,two folding nucleotide events were clearly identified. This novel approach uses solely the feature space made of the mean ionic current and the ionic current blockade height, while avoiding sparse patterns obtained from a blockade level or event analysis of the raw data using the dwell time feature. The proposal for the relevance and importance of the ionic blockade height was supported by the fact that this feature can also cope with data set sparsity. The blockade height is proven efficient in all cases where the traditional dwell time fails to identify clusters of events.

The novel feature selection proposed here together with unsupervised learning can be valuable in providing a focal point on the 2D nanopore experimental data and a valuable insight into the nanopore. The beginning here is the identification of molecular events in the pore. Having shown the applicability of this scheme to DNA, it can further extended for other types of biomolecules, which could involve a more rich conformational variation. These rich dynamics is an aspect implicitly carried within the ionic current blockade height feature. In order to reveal the exact configurational aspects, more data are needed. Overall, though, this framework can also be used with data from other nanopores in order to gain insight into the differences among nanopores and guide the design of the nanopore material. Overall, the relevance of the proposed scheme, in view of sensing DNA and also detecting its sequences is very high. An efficient 2D nanopore should be able not only to detect the molecule, but also read-out the order of its nucleobases with the lowest possible errors. These errors arise from different sources and are often difficult to control or reduce. The classification scheme based on using the blockade height as a feature can be further tested, improved, and applied to other types of DNA sequences in order to reduce errors and filter out the information on the identity of the nucleobase along a translocating DNA. Training with small datasets returned high accuracy values. The classification scheme has the potential to deal with different nanopore materials and sizes, as well as varying experimental conditions (applied voltage, salt concentration, pH, etc.). In this respect, the highest gain of our approach would be to be used in generating a database of 2D nanopore events inherently including all the above details. This can in turn be valuable not only for interpreting events, but also predicting DNA sequences in 2D nanopore experiments. This work provides an important methodological input towards this direction. In the end, the importance of this work is two-fold: it provides an efficient framework for interpreting experimental nanopore data, while it can also deliver a feedback to the experiments for tuning and optimizing the signals in view of biomolecule identifying and sequencing.

Acronyms

CCE	Categorical Cross Entropy
CH-score	Calinski-Harabasz Score
CNN	Convolutional Neural Network
CUSUM	Cumulative sum algorithm
dAMP	Deoxyadenosine monophosphate
dCMP	Deoxycytidine monophosphate
dGMP	Deoxyguanosine monophosphate
DL	Deep Learning
DNA	Deoxyribonucleic acid
DNN	Deep neural Network
dTMP	Deoxythymidine monophosphate
HL	Hidden Layer
ML	Machine Learning
RF	Random Forest
RNN	Recurrent Neural Network
S-score	Silhouette Score

Bibliography

1. M. Fyta, S. Melchionna, and S. Succi, "Translocation of biomolecules through solid-state nanopores: theory meets experiments", *Journal of Polymer Science Part B: Polymer Physics*, vol. 49:no. 14, pp. 985–1011, 2011. DOI: 10.1002/polb.22284. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/polb.22284. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/polb.22284.
2. R. W. DeBlois and C. P. Bean, "Counting and sizing of submicron particles by the resistive pulse technique", *Review of Scientific Instruments*, vol. 41:no. 7, pp. 909–916, 1970. DOI: 10.1063/1.1684724. eprint: https://doi.org/10.1063/1.1684724. [Online]. Available: https://doi.org/10.1063/1.1684724.
3. D. Branton, D. W. Deamer, A. Marziali, H. Bayley, S. A. Benner, T. Butler, M. Di Ventra, S. Garaj, A. Hibbs, X. Huang, S. B. Jovanovich, P. S. Krstic, S. Lindsay, X. S. Ling, C. H. Mastrangelo, A. Meller, J. S. Oliver, Y. V. Pershin, J. M. Ramsey, R. Riehn, G. V. Soni, V. Tabard-Cossa, M. Wanunu, and M. W. amd Jeffery A Schloss, "The potential and challenges of nanopore sequencing", *Nature Biotechnol.*, vol. 26:no. 10, pp. 1146–1153, 2008.
4. C. Dekker, "Solid-state nanopores", *Nature Nanotech.*, vol. 2:no. 4, pp. 209–215, 2007.
5. M. Fyta, "Threading dna through nanopores for biosensing applications", *J. Physics: Condens. Matter*, vol. 27:no. 27, p. 273 101, 2015.
6. J. Li, D. Stein, C. McMullan, D. Branton, M. J. Aziz, and J. A. Golovchenko, "Ion-beam sculpting at nanometre length scales", *Nature*, vol. 412:no. 6843, pp. 166–169, 2001.
7. D. Fologea, M. Gershow, B. Ledden, D. S. McNabb, J. A. Golovchenko, and J. Li, "Detecting single stranded dna with a solid state nanopore", *Nano Lett.*, vol. 5:no. 10, pp. 1905–1909, 2005.
8. Z. Zhou, Y. Hu, H. Wang, Z. Xu, W. Wang, X. Bai, X. Shan, and X. Lu, "Dna translocation through hydrophilic nanopore in hexagonal boron nitride", *Sci. Rep.*, vol. 3, p. 3287, 2013.
9. D. Fologea, J. Uplinger, B. Thomas, D. S. McNabb, and J. Li, "Slowing dna translocation in a solid-state nanopore", *Nano Lett.*, vol. 5:no. 9, pp. 1734–1737, 2005.

10. K. Liu, J. Feng, A. Kis, and A. Radenovic, “Atomically thin molybdenum disulfide nanopores with high sensitivity for dna translocation”, *ACS Nano*, vol. 8:no. 3, pp. 2504–2511, 2014.
11. J. Feng, K. Liu, R. D. Bulushev, S. Khlybov, D. Dumcenco, A. Kis, and A. Radenovic, “Identification of single nucleotides in mos$_2 nanopores$”, *Nature Nanotech.*, vol. 10:no. 12, p. 1070, 2015.
12. S. Garaj, W Hubbard, A Reina, J Kong, D Branton, and J. Golovchenko, “Graphene as a subnanometre trans-electrode membrane”, *Nature*, vol. 467:no. 7312, p. 190, 2010.
13. C. A. Merchant, K. Healy, M. Wanunu, V. Ray, N. Peterman, J. Bartel, M. D. Fischbein, K. Venta, Z. Luo, A. C. Johnson, and M. Drndić, “Dna translocation through graphene nanopores”, *Nano Lett.*, vol. 10:no. 8, pp. 2915–2921, 2010.
14. G. F. Schneider, S. W. Kowalczyk, V. E. Calado, G. Pandraud, H. W. Zandbergen, L. M. Vandersypen, and C. Dekker, “Dna translocation through graphene nanopores”, *Nano Lett.*, vol. 10:no. 8, pp. 3163–3167, 2010.
15. Z. S. Siwy and M. Davenport, “Nanopores: graphene opens up to dna”, *Nature Nanotech.*, vol. 5:no. 10, p. 697, 2010.
16. M. Wanunu, “Nanopores: a journey towards dna sequencing”, *Phys. Life Rev.*, vol. 9:no. 2, pp. 125–158, 2012.
17. H. Bayley, “Nanotechnology: holes with an edge”, *Nature*, vol. 467:no. 7312, p. 164, 2010.
18. S. W. Kowalczyk, A. Y. Grosberg, Y. Rabin, and C. Dekker, “Modeling the conductance and dna blockade of solid-state nanopores”, *Nanotech.*, vol. 22:no. 31, p. 315 101, 2011.
19. D. W. Deamer and M. Akeson, “Nanopores and nucleic acids: prospects for ultrarapid sequencing”, *Trends in Biotech.*, vol. 18:no. 4, pp. 147–151, 2000.
20. K. Venta, G. Shemer, M. Puster, J. A. Rodriguez-Manzo, A. Balan, J. K. Rosenstein, K. Shepard, and M. Drndić, “Differentiation of short, single-stranded dna homopolymers in solid-state nanopores”, *ACS Nano*, vol. 7:no. 5, pp. 4629–4636, 2013.
21. M. Wanunu, J. Sutin, B. McNally, A. Chow, and A. Meller, “Dna translocation governed by interactions with solid-state nanopores”, *Biophys. J.*, vol. 95:no. 10, pp. 4716–4725, 2008.
22. J. Mathé, H. Visram, V. Viasnoff, Y. Rabin, and A. Meller, “Nanopore unzipping of individual dna hairpin molecules”, *Biophys. J.*, vol. 87:no. 5, pp. 3205–3212, 2004.

23. A. Meller, L. Nivon, and D. Branton, "Voltage-driven dna translocations through a nanopore", *Phys. Rev. Lett.*, vol. 86:no. 15, p. 3435, 2001.

24. S. Benner, R. J. Chen, N. A. Wilson, R. Abu-Shumays, N. Hurt, K. R. Lieberman, D. W. Deamer, W. B. Dunbar, and M. Akeson, "Sequence-specific detection of individual dna polymerase complexes in real time using a nanopore", *Nature Nanotech.*, vol. 2:no. 11, p. 718, 2007.

25. A. Meller, L. Nivon, E. Brandin, J. Golovchenko, and D. Branton, "Rapid nanopore discrimination between single polynucleotide molecules", *Proc. Nat. Ac. Sci.*, vol. 97:no. 3, pp. 1079–1084, 2000.

26. J. Li, M. Gershow, D. Stein, E. Brandin, and J. A. Golovchenko, "Dna molecules and configurations in a solid-state nanopore microscope", *Nat. Mater.*, vol. 2:no. 9, p. 611, 2003.

27. M. Wanunu, T. Dadosh, V. Ray, J. Jin, L. McReynolds, and M. Drndić, "Rapid electronic detection of probe-specific micrornas using thin nanopore sensors", *Nat. Nanotech.*, vol. 5:no. 11, p. 807, 2010.

28. M. Zwolak and M. Di Ventra, "Electronic signature of dna nucleotides via transverse transport", *Nano Lett.*, vol. 5:no. 3, pp. 421–424, 2005.

29. B. McNally, A. Singer, Z. Yu, Y. Sun, Z. Weng, and A. Meller, "Optical recognition of converted dna nucleotides for single-molecule dna sequencing using nanopore arrays", *Nano Lett.*, vol. 10:no. 6, pp. 2237–2244, 2010.

30. M. E. Gracheva, A. Xiong, A. Aksimentiev, K. Schulten, G. Timp, and J.-P. Leburton, "Simulation of the electric response of dna translocation through a semiconductor nanopore–capacitor", *Nanotech.*, vol. 17:no. 3, p. 622, 2006.

31. R. M. Smeets, U. F. Keyser, N. H. Dekker, and C. Dekker, "Noise in solid-state nanopores", *Proc. Nat. Ac. Sci.*, vol. 105:no. 2, pp. 417–421, 2008.

32. A. Balan, B. Machielse, D. Niedzwiecki, J. Lin, P. Ong, R. Engelke, K. L. Shepard, and M. Drndić, "Improving signal-to-noise performance for dna translocation in solid-state nanopores at mhz bandwidths", *Nano Lett.*, vol. 14:no. 12, pp. 7215–7220, 2014.

33. J. Schreiber, Z. L. Wescoe, R. Abu-Shumays, J. T. Vivian, B. Baatar, K. Karplus, and M. Akeson, "Error rates for nanopore discrimination among cytosine, methylcytosine, and hydroxymethylcytosine along individual dna strands", *Proc. Nat. Ac. Sci.*, vol. 110:no. 47, pp. 18 910–18 915, 2013. DOI: 10.1073/pnas.1310615110.

34. M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The oxford nanopore minion: delivery of nanopore sequencing to the genomics community", *Genome biology*, vol. 17:no. 1, p. 239, 2016.

35. M.-A. Madoui, S. Engelen, C. Cruaud, C. Belser, L. Bertrand, A. Alberti, A. Lemainque, P. Wincker, and J.-M. Aury, “Genome assembly using nanopore-guided long and error-free dna reads”, *BMC Genomics*, vol. 16:no. 1, p. 327, 2015.

36. V. Boža, B. Brejová, and T. Vinař, “Deepnano: deep recurrent neural networks for base calling in minion nanopore reads”, *PloS One*, vol. 12:no. 6, e0178751, 2017.

37. Y. Li, R. Han, C. Bi, M. Li, S. Wang, and X. Gao, “Deepsimulator: a deep simulator for nanopore sequencing”, *Bioinformatics*, vol. 1, p. 10, 2018.

38. P. M. Ashton, S. Nair, T. Dallman, S. Rubino, W. Rabsch, S. Mwaigwisya, J. Wain, and J. O’grady, “Minion nanopore sequencing identifies the position and structure of a bacterial antibiotic resistance island”, *Nat. Biotech.*, vol. 33:no. 3, p. 296, 2015.

39. C. Raillon, P. Granjon, M. Graf, L. J. Steinbock, and A. Radenovic, “Fast and automatic processing of multi-level events in nanopore translocation experiments”, *Nanoscale*, vol. 4, pp. 4916–4924, 16 2012.

40. J. M. Schreiber and K. Karplus, “Segmentation of noisy signals generated by a nanopore”, *bioRxiv*, p. 014 258, 2015. DOI: 10.1101/014258.

41. M. Mihovilovic, N. Hagerty, and D. Stein, “Statistics of dna capture by a solid-state nanopore”, *Phys. Rev. Lett.*, vol. 110, p. 028 102, 2 2013. DOI: 10.1103/PhysRevLett.110.028102.

42. C. R. Collins, G. J. Gordon, O. A. von Lilienfeld, and D. J. Yaron, “Constant size descriptors for accurate machine learning models of molecular properties”, *J. Chem. Phys.*, vol. 148:no. 24, p. 241 718, 2018. DOI: 10.1063/1.5020441.

43. Z. L. Wescoe, J. Schreiber, and M. Akeson, “Nanopores discriminate among five c5-cytosine variants in dna”, *J. Am. Chem. Soc.*, vol. 136, pp. 16 582–16 587, 2014.

44. J. Schreiber, Z. L. Wescoe, R. Abu-Shumays, J. T. Vivian, B. Baatar, K. Karplus, and M. Akeson, “Error rates for nanopore discrimination among cytosine, methylcytosine, and hydroxymethylcytosine along individual dna strands”, *Proc. Nat. Ac. Sci.*, vol. 110:no. 47, pp. 18 910–18 915, 2013.

45. Y. Zhang, L. Liu, J. Sha, Z. Ni, H. Yi, and Y. Chen, “Nanopore detection of dna molecules in magnesium chloride solutions.”, *Nanoscale Res. Lett.*, vol. 8, p. 245, 2013.

46. C. Marchet, L. Lecompte, C. D. Silva, C. Cruaud, J.-M. Aury, J. Nicolas, and P. Peterlongo, “De novo clustering of long reads by gene from transcriptomics data”, *Nucleic acids research*, vol. 47:no. 1, e2–e2, 2018.

47. M. Krachunov, M. Nisheva, and D. Vassilev, “Machine learning models in error and variant detection in high-variation high-throughput sequencing datasets”, *Procedia Comput. Sci.*, vol. 108, pp. 1145–1154, 2017. DOI: https://doi.org/10.1016/j.procs.2017.05.242.

48. L. B. Holder, M. M. Haque, and M. K. Skinner, “Machine learning for epigenetics and future medical applications”, *Epigenetics*, vol. 12:no. 7, pp. 505–514, 2017. DOI: 10.1080/15592294.2017.1329068.

49. T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, A. Gitter, and C. S. Greene, “Opportunities and obstacles for deep learning in biology and medicine”, *J. R. Soc. Interface*, vol. 15, p. 20 170 387, 2018.

50. R. Sanchez-Garcia, C. O. S. Sorzano, J. M. Carazo, and J. Segura, “3dcons-db: a database of position-specific scoring matrices in protein structures”, *Molecules*, vol. 22, p. 2230, 2017.

51. A. Barati Farimani, M. Heiranian, and N. R. Aluru, “Identification of amino acids with sensitive nanoporous mos2: towards machine learning-based prediction”, *NPJ 2D Mater. Appl.*, vol. 2:no. 1, p. 14, 2018. DOI: 10.1038/s41699-018-0060-8.

52. M. Kolmogorov, E. Kennedy, Z. Dong, G. Timp, and P. A. Pevzner, “Single-molecule protein identification by sub-nanopore sensors”, *PLOS Comput. Biol.*, vol. 13, 2017.

53. Q. Zhou and J. S. Liu, “Extracting sequence features to predict protein–dna interactions: a comparative study”, *Nucleic Acids Res.*, vol. 36:no. 12, pp. 4137–4148, 2008. DOI: 10.1093/nar/gkn361.

54. P. Krstić, B. Ashcroft, and S. Lindsay, “Physical model for recognition tunneling”, *Nanotech.*, vol. 26:no. 8, p. 084 001, 2015.

55. M. Stoiber and J. Brown, “Basecrawller: streaming nanopore basecalling directly from raw signal”, *bioRxiv*, p. 133 058, 2017.

56. M. Landry and S. Winters-Hilt, “Analysis of nanopore detector measurements using machine-learning methods, with application to single-molecule kinetic analysis”, *BMC Bioinformatics*, vol. 8:no. 7, S12, 2007. DOI: 10.1186/1471-2105-8-S7-S12.

57. A. Churbanov and S. Winters-Hilt, “Clustering ionic flow blockade toggles with a mixture of hmms”, *BMC Bioinformatics*, vol. 9:no. 9, S13, 2008. DOI: 10.1186/1471-2105-9-S9-S13.

58. J. Schreiber and K. Karplus, “Analysis of nanopore data using hidden markov models”, *Bioinformatics*, vol. 31:no. 12, pp. 1897–1903, 2015. DOI: 10.1093/bioinformatics/btv046.

59. S. Winters-Hilt and C. Baribault, "A novel, fast, hmm-with-duration implementation - for application with a new, pattern recognition informed, nanopore detector", *BMC Bioinformatics*, vol. 8 Suppl 7:no. 7, S19–S19, 2007. DOI: 10.1186/1471-2105-8-S7-S19.

60. H. Teng, M. D. Cao, M. B. Hall, T. Duarte, S. Wang, and L. J. M. Coin, "Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning", *GigaScience*, vol. 7:no. 5, pp. 1–10, 2018. DOI: 10.1093/gigascience/giy037.

61. K. Misiunas, N. Ermann, and U. F. Keyser, "Quipunet: convolutional neural network for single-molecule nanopore sensing", *Nano Lett.*, vol. 18:no. 6, pp. 4040–4045, 2018. DOI: 10.1021/acs.nanolett.8b01709.

62. R. Luo, F. J. Sedlazeck, T.-W. Lam, and M. Schatz, "Clairvoyante: a multi-task convolutional deep neural network for variant calling in single molecule sequencing", *bioRxiv*, p. 310 458, 2018.

63. A. D. Carral, C. S. Sarap, K. Liu, A. Radenovic, and M. Fyta, "2d MoS2 nanopores: ionic current blockade height for clustering DNA events", *2D Materials*, vol. 6:no. 4, p. 045 011, 2019. DOI: 10.1088/2053-1583/ab2c38. [Online]. Available: https://doi.org/10.1088%2F2053-1583%2Fab2c38.

64. T. A. Waigh, "A molecular approach for physical scientistis", in *Applied Biophysics*. John Wiley Sons, Ltd, 2007, pp. 407–421, ISBN: 9780470513156. DOI: 10.1002/9780470513156.index.

65. J. D. WATSON and F. H. C. CRICK, "Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid", *Nature*, vol. 171:no. 4356, pp. 737–738, 1953. DOI: 10.1038/171737a0. [Online]. Available: https://doi.org/10.1038/171737a0.

66. F. H. C. Crick, J. D. Watson, and W. L. Bragg, "The complementary structure of deoxyribonucleic acid", *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 223:no. 1152, pp. 80–96, 1954. DOI: 10.1098/rspa.1954.0101. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1954.0101. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1954.0101.

67. R. E. Hardisty, "Methods to probe the function of modified bases in dna (doctoral thesis)", *University of Cambridge*, 2017. DOI: https://doi.org/10.17863/CAM.18804.

68. S. Freeman, *Fundamentos de biología*, ser. Fuera de colección Out of series. Pearson Educación, 2010, ISBN: 9788478291212. [Online]. Available: https://books.google.de/books?id=6Au6XwAACAAJ.

69. R. Hagman, "Probing the complex between the factor for inversion stimulation and dna using a nanofluidic device. (proceedings)", 2013.

70. F. CRICK, "Central dogma of molecular biology", *Nature*, vol. 227:no. 5258, pp. 561–563, 1970. DOI: 10.1038/227561a0. [Online]. Available: https://doi.org/10.1038/227561a0.

71. W. Commons, *File:main nucleobases.png — wikimedia commons, the free media repository*, [Online; accessed 8-September-2019], 2016. [Online]. Available: https://commons.wikimedia.org/w/index.php?title=File:Main_nucleobases.png&oldid=226753854.

72. ——, *File:dna structure+key+labelled.pn nobb.png — wikimedia commons, the free media repository*, [Online; accessed 8-September-2019], 2019. [Online]. Available: \url{https://commons.wikimedia.org/w/index.php?title=File:DNA_Structure%2BKey%2BLabelled.pn_NoBB.png&oldid=364264502}.

73. J. Thomas Simpson, "Efficient sequence assembly andvariant calling using compressed data structures (doctoral thesis)", *University of Cambridge*, 2012.

74. A. M. Maxam and W Gilbert, "A new method for sequencing dna", *Proceedings of the National Academy of Sciences*, vol. 74:no. 2, pp. 560–564, 1977, ISSN: 0027-8424. DOI: 10.1073/pnas.74.2.560. eprint: https://www.pnas.org/content/74/2/560.full.pdf. [Online]. Available: https://www.pnas.org/content/74/2/560.

75. A. Munshi, *DNA Sequencing-Methods and Applications*, 1st. InTech, 2012, ISBN: 978-953-51-0564-0.

76. F. Sanger, G. M. Air, B. G. Barrell, N. L. Brown, A. R. Coulson, J. C. Fiddes, C. A. Hutchison, P. M. Slocombe, and M. Smith, "Nucleotide sequence of bacteriophage x174 dna", *Nature*, vol. 265:no. 5596, pp. 687–695, 1977. DOI: 10.1038/265687a0. [Online]. Available: https://doi.org/10.1038/265687a0.

77. J. Siqueira José F, A. F. Fouad, and I. N. Rôças, "Pyrosequencing as a tool for better understanding of human microbiomes", *Journal of oral microbiology*, vol. 4, 10.3402/jom.v4i0.10743, 2012. DOI: 10.3402/jom.v4i0.10743. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/22279602.

78. M. Ratkovíc, "Deep learning model for base calling of minion nanopore reads (masther thesis)", *University of Zagreb*, 2017.

79. M. V. Olson, "The human genome project", *Proceedings of the National Academy of Sciences of the United States of America*, vol. 90:no. 10, pp. 4338–4344, 1993. DOI: 10.1073/pnas.90.10.4338. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/8506271.

80. S. Sonnenburg, "Machine learning for genomic sequence analysis (doctoral thesis)", *Technische Universität Berlin*, 2009. DOI: 10.14279/depositonce-2055.

81. C.-X. Song, T. A. Clark, X.-Y. Lu, A. Kislyuk, Q. Dai, S. W. Turner, C. He, and J. Korlach, "Sensitive and specific single-molecule sequencing of 5-hydroxymethylcytosine", *Nature methods*, vol. 9:no. 1, pp. 75–77, 2011. DOI: `10.1038/nmeth.1779`. [Online]. Available: `https://www.ncbi.nlm.nih.gov/pubmed/22101853`.

82. N. Varongchayakul, J. Song, A. Meller, and M. W. Grinstaff, "Single-molecule protein sensing in a nanopore: a tutorial", *Chem. Soc. Rev.*, vol. 47, pp. 8512–8524, 23 2018. DOI: `10.1039/C8CS00106E`. [Online]. Available: `http://dx.doi.org/10.1039/C8CS00106E`.

83. H. Lu, F. Giordano, and Z. Ning, "Oxford nanopore minion sequencing and genome assembly", *Genomics, Proteomics Bioinformatics*, vol. 14:no. 5, pp. 265 –279, 2016, SI: Big Data and Precision Medicine, ISSN: 1672-0229. DOI: `https://doi.org/10.1016/j.gpb.2016.05.004`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S1672022916301309`.

84. Z. Liu, Y. Wang, T. Deng, and Q. Chen, "Solid-state nanopore-based dna sequencing technology", *Journal of Nanomaterials*, vol. 2016, pp. 1–13, 2016. DOI: `10.1155/2016/5284786`.

85. W. Si, Y. Zhang, J. Sha, and Y. Chen, "Controllable and reversible dna translocation through a single-layer molybdenum disulfide nanopore", *Nanoscale*, vol. 10, pp. 19 450–19 458, 41 2018. DOI: `10.1039/C8NR05830J`. [Online]. Available: `http://dx.doi.org/10.1039/C8NR05830J`.

86. O. Simeone, "A very brief introduction to machine learning with applications to communication systems", *IEEE Transactions on Cognitive Communications and Networking*, vol. 4:no. 4, pp. 648–664, 2018. DOI: `10.1109/TCCN.2018.2881442`.

87. P. V. L. Martins, "Gene prediction using deep learning (master thesis)", *Universidade do Porto*, 2018.

88. S. B. Kotsiantis, "Supervised machine learning: a review of classification techniques", in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2007, pp. 3–24, ISBN: 978-1-58603-780-2. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1566770.1566773`.

89. X. Zhu, "Semi-supervised learning with graphs (doctoral thesis)", *Carnegie Mellon University*, 2005, AAI3179046.

90. D. Greene, P. Cunningham, and R. Mayer, "Unsupervised learning and clustering", *Lecture Notes in Applied and Computational Mechanics*, pp. 51–90, 2008. DOI: `10.1007/978-3-540-75171-7-3`.

91. O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning*, 1st. The MIT Press, 2010, ISBN: 0262514125, 9780262514125.

92. S. Becker and M. D. Plumbley, "Unsupervised neural network learning procedures for feature extraction and classification", *Applied Intelligence*, vol. 6, pp. 185–203, 1996.

93. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

94. L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: a survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996. [Online]. Available: http://people.csail.mit.edu/lpk/papers/rl-survey.ps.

95. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988, ISBN: 0-13-022278-X.

96. J. Han, M. Kamber, and A. K. H. Tung, "Spatial clustering methods in data mining: a survey", in *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS*, H. J. Miller and J. Han, Eds., Taylor and Francis, 2001. [Online]. Available: http://www-faculty.cs.uiuc.edu/~hanj/pdf/gkdbk01.pdf.

97. J. J. Shen, P.-H. Lee, J. J. A. Holden, and H. Shatkay, "Using cluster ensemble and validation to identify subtypes of pervasive developmental disorders", *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2007, pp. 666–670, 2007. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/18693920.

98. T. Xięski, A. Nowak-Brzezińska, and A. Wakulicz-Deja, "Density-based method for clustering and visualization of complex data", in *Rough Sets and Current Trends in Computing*, J. Yao, Y. Yang, R. Słowiński, S. Greco, H. Li, S. Mitra, and L. Polkowski, Eds., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 142–149, ISBN: 978-3-642-32115-3.

99. I. MR and D. MOHAN, "A survey of grid based clustering algorithms", *International Journal of Engineering Science and Technology*, vol. 2, 2010.

100. A. Zhang, "Acceleration of k-means clustering by k-dijkstra method for graph partitioning (master thesis)", 2015, AAI3179046.

101. T. Thinsungnoen, N. Kaoungku, P. Durongdumronchai, K. Kerdprasop, and N. Kerdprasop, "The clustering validity with silhouette and sum of squared errors (proceedings)", 2015, pp. 44–51. DOI: 10.12792/iciae2015.012.

102. P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005, ISBN: 0321321367.

103. B. G. Farley and W. A. Clark, "Simulation of self-organizing systems by digital computer", *Trans. of the IRE Professional Group on Information Theory (TIT)*, vol. 4, pp. 76–84, 1954.

104. C. von der Malsburg, "Frank rosenblatt: principles of neurodynamics: perceptrons and the theory of brain mechanisms", *Brain Theory*, pp. 245–248, 1986. DOI: 10.1007/978-3-642-70911-1_20.

105. C. Münker, "Using convolutional neural networks to distinguish vehicle pose and vehicle class (master thesis)", *Technische Universität Darmstadt*, 2016, AAI3179046.

106. D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex", *The Journal of physiology*, vol. 160:no. 1, pp. 106–154, 1962. DOI: 10.1113/jphysiol.1962.sp006837. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/14449617.

107. C. Angermüller, "Deep neural networks and statistical models for studying single-cell dna methylation (doctoral thesis", *European Bioinformatics Institute, University of Cambridge. Trinity Hall.*, 2017. DOI: 10.14279/depositonce-2055.

108. L. A. Santamaría, S. Zuñiga, I. H. Pineda, M. J. Somodevilla, and M. Rossainz, "Reconocimiento de genes en secuencias de adn por medio de imágenes.".

109. M. R. Ruska, "Using machine learning to predict and better understand dna methylation and genomic enhancers (doctoral thesis)", *Freie Universität Berlin*, 2018.

110. S. Chatterjee, A. Iyer, S. Avva, A. Kollara, and M. Sankarasubbu, "Convolutional Neural Networks In Classifying Cancer Through DNA Methylation", *arXiv e-prints*, arXiv:1807.09617, arXiv:1807.09617, 2018. arXiv: 1807.09617 [q-bio.GN].

111. V. Boza, B. Brejova, and T. Vinar, "Deepnano: deep recurrent neural networks for base calling in minion nanopore reads", *PLoS ONE*, vol. 12:no. 6, pp. 1–13, 2017.

112. Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions", *arXiv e-prints*, vol. abs/1605.02688, 2016. [Online]. Available: http://arxiv.org/abs/1605.02688.

113. M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, "Nanopore sequencing and assembly of a human genome with ultra-long reads", *Nat. Biotechnol.*, vol. 36, 338 EP –, 2018. [Online]. Available: https://doi.org/10.1038/nbt.4060.

114. M. David, L. J. Dursi, D. Yao, P. C. Boutros, and J. T. Simpson, "Nanocall: an open source basecaller for oxford nanopore sequencing data", *Bioinformatics*, vol. 33:no. 1, pp. 49–55, 2016.

115. M. Santos, *Nucleic Acids from A to Z: A Concise Encyclopedia*. 2008, ISBN: 978-0-470-08012-2.

116. Y. Feng, Y. Zhang, C. Ying, D. Wang, and C. Du, "Nanopore-based fourth-generation dna sequencing technology", *Genomics Proteomics Bioinformatics*, vol. 13:no. 1, pp. 4–16, 2015.

117. M. E. Celebi and K. Aydin, *Unsupervised Learning Algorithms*, 1st. Springer Publishing Company, Incorporated, 2016, ISBN: 3319242091, 9783319242095.

118. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: machine learning in Python", *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

119. J. Baarsch and M. Emre Celebi, "Investigation of internal validity measures for k-means clustering", *Proc. Int. MultiConference of Eng. Comput. Sci.*, vol. 1, 2012.

120. T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24:no. 7, pp. 881–892, 2002. DOI: 10.1109/TPAMI.2002.1017616.

121. P. C. Faucon, R. Trevino, P. Balachandran, K. Standage-Beier, and X. Wang, "High accuracy base calls in nanopore sequencing", *bioRxiv*, p. 126 680, 2017. DOI: 10.1101/126680.

122. P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis", *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

123. C. Cengizler and M. Kerem Un, "Evaluation of calinski-harabasz criterion as fitness measure for genetic algorithm based segmentation of cervical cell nuclei", *British J. Math. Comput. Sci.*, vol. 22:no. 6, pp. 1–13, 2017.

124. Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu, "Understanding of internal clustering validation measures", *IEEE International Conference on Data Mining*, pp. 911–916, 2010.

125. F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

126. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: large-scale machine*

learning on heterogeneous systems, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

127. G. Aoki and Y. Sakakibara, "Convolutional neural networks for classification of alignments of non-coding rna sequences", *Bioinformatics (Oxford, England)*, vol. 34, pp. i237–i244, 2018. DOI: 10.1093/bioinformatics/bty228.

128. I. Mohamad and D. Usman, "Standardization and its effects on k-means clustering algorithm", vol. 6, 2013, pp. 3299–3303.

129. Y. Jain and S. Bhandare, "Min max normalization based data perturbation method for privacy protection", *International Journal of Computer*, vol. 2, 2011.

130. M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, and E. Kaxiras, "Quantized current blockade and hydrodynamic correlations in biopolymer translocation through nanopores: evidence from multiscale simulations", *Nano Lett.*, vol. 8:no. 4, pp. 1115–1119, 2008.

131. D. Kingma and J. Ba, "Adam: a method for stochastic optimization", *International Conference on Learning Representations*, 2014.

132. P. Boer, D. Kroese, S. Mannor, and R. Rubinstein, "A tutorial on the cross-entropy method", *Ann Oper Res*, vol. 134, 2002.

133. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

Codes

Clustering Analysis	14 pages of clustering python code
Nucleotide Prediction	6 pages of nucleotide prediction python code
Feature Prediction	3 pages of feature prediction python code

```
# -*- coding: utf-8 -*-
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import pylab
from collections import Counter
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_blobs
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm

import os
os.chdir("/Users/ANGEL/Desktop/HiWi/Data Aleksandra")
#-----

mat = scipy.io.loadmat('100mv_2.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(45)
fits = mat['ConcatenatedFits']
evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(45)

ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(45):
    coord[i] = np.asscalar(crd[0,i])
    #evfitt[i] = np.asscalar(evfit[0,i])

# print(np.size(evfit[0,11]))
# print(coord[11])
# #plt.plot(ev[coord[11]:coord[11]+np.size(evfit[0,11])])
# plt.plot(ev[coord[11]+173+17:coord[11]+173+17+18])
# plt.show()


## TIME ##
time = np.ndarray(45)
```

```
for i in range(45):
    time[i] = np.size(evfit[0,i])

## HEIGHT ##
h = np.ndarray(45)
for i in range(45):
    #h[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i]))]
## MEAN ##
mu = np.ndarray(45)
for i in range(45):
    mu[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(evfi

# ###∂∫∫∂å∫
# aa=0
# muprb = np.zeros(79654)
# for i in range (79653):
#     indx = coord[aa]+(np.size(evfit[0,aa]))/2
#     if fits[i] == fits[i+1]:
#         muprb[coord[aa]:coord[aa]+np.size(evfit[0,aa])] = mu[aa
#
#     aa += 1
## NLEVELS ##
lvl = mat['EventDatabase']
lvl = lvl['NumberOfLevels']
lvls = np.ndarray(45)
for i in range(45):
    lvls[i] = np.asscalar(lvl[0,i])

## LEVELS ##
lev = mat['EventDatabase']
lev = lev['Levels']
levs = np.ndarray(45)
levss = np.ndarray(45)
for i in range(45):
    #levs[i] = np.asscalar(lev[0,i])
    a = np.size(lev[0,i])/6

        #levss[i] = np.asscalar(levs[0,i])
## COULOMB##
cou = mat['EventDatabase']
cou = cou['AreaCoulomb']
acou = np.ndarray(45)
for i in range(45):
    acou[i] = np.asscalar(cou[0,i])
```

```
# plotting features 2x2 ##
# time = (time - time.mean()) / time.std()
# mu = (mu - mu.mean()) / mu.std()
# h = (h - h.mean()) / h.std()
xx = np.arange(45)
# plt.plot(time,mu,'+')
# plt.show()
#plt.scatter(np.arange(45),sorted(mu))
#plt.show()
## 3dplot ##

# ax = plt.axes(projection='3d')
# ax.scatter3D(time, mu, h, c=time, cmap='viridis')
# plt.xlabel("I Mean ", fontsize=15);
# plt.ylabel("Height", fontsize=15);
# plt.show()
# plt.savefig('100mv_2pl')


# time = (time - time.mean()) / time.std()
# mu = (mu - mu.mean()) / mu.std()
# h = (h - h.mean()) / h.std()
#data = np.ndarray(shape=(np.size(index),2))
#data[:,0] = mu[index]
#data[:,1] = time[index]

# K Means #
# time = (time - time.mean())#/ time.std()
# mu = (mu - mu.mean()) #/ mu.std()
# h = (h - h.mean()) #/ h.std()

#data[:,2] = mu
# db = DBSCAN(eps=0.5, min_samples=2).fit(data)
# core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
# core_samples_mask[db.core_sample_indices_] = True
# labels = db.labels_
#
# # Number of clusters in labels, ignoring noise if present.
# n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
#
# print('Estimated number of clusters: %d' % n_clusters_)

#d = km.cluster_centers_
#print(d)
```

```
############################

mat = scipy.io.loadmat('100mv_1.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(31)

evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(31)

ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(31):
    coord[i] = np.asscalar(crd[0,i])
    #evfitt[i] = np.asscalar(evfit[0,i])
#
# # print(np.size(evfit[0,1]))
# # print(coord[0,1])
# # plt.plot(ev[coord[0,22]:coord[0,22]+np.size(evfit[0,22])])
# # plt.show()
#
## TIME ##
time2 = np.ndarray(31)
for i in range(31):
    time2[i] = np.size(evfit[0,i])

## HEIGHT ##
h2 = np.ndarray(31)
for i in range(31):
    #h2[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h2[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i]))
## MEAN ##
mu2 = np.ndarray(31)
for i in range(31):
    mu2[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(ev
    ## NLEVELS ##
lvl2 = mat['EventDatabase']
lvl2 = lvl2['NumberOfLevels']
lvls2 = np.ndarray(31)
for i in range(31):
    lvls2[i] = np.asscalar(lvl2[0,i])
#
#
```

```
##plotting features 2x2 ##
# time2 = (time2 - time2.mean()) #/ time2.std()
# mu2 = (mu2 - mu2.mean()) #/ mu2.std()
# h2 = (h2 - h2.mean()) #/ h2.std()

# plt.plot(time,-mu,'+')
# plt.xlabel("Dwell time", fontsize=15);
# plt.ylabel("Mean", fontsize=15);
# plt.show()

# ## 3dplot ##
# ax = plt.axes(projection='3d')
# ax.scatter3D(mu, h, time, c=time, cmap='viridis')
# plt.xlabel("I Mean ", fontsize=15);
# plt.ylabel("Height", fontsize=15);
# #plt.show()
# plt.savefig('143303pl')
mat = scipy.io.loadmat('50mv_selected.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(13)

evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(13)

ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(13):
    coord[i] = np.asscalar(crd[0,i])
    #evfitt[i] = np.asscalar(evfit[0,i])
#
# # print(np.size(evfit[0,1]))
# # print(coord[0,1])
# # plt.plot(ev[coord[0,22]:coord[0,22]+np.size(evfit[0,22])])
# # plt.show()
#
## TIME ##
time3 = np.ndarray(13)
for i in range(13):
    time3[i] = np.size(evfit[0,i])

## HEIGHT ##
h3 = np.ndarray(13)
```

```
for i in range(13):
    #h3[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h3[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i])
## MEAN ##
mu3 = np.ndarray(13)
for i in range(13):
    mu3[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(ev

    ## NLEVELS ##
lvl3 = mat['EventDatabase']
lvl3 = lvl3['NumberOfLevels']
lvls3 = np.ndarray(13)
for i in range(13):
    lvls3[i] = np.asscalar(lvl3[0,i])

#######################
########################
######################
######################
######################

os.chdir("/Users/ANGEL/Desktop/HiWi/Data Aleksandra/Homopolymers"

mat = scipy.io.loadmat('A1.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(np.size(crd))
fits = mat['ConcatenatedFits']
evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(np.size(crd))

ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(np.size(crd)):
    coord[i] = np.asscalar(crd[0,i])
#evfitt[i] = np.asscalar(evfit[0,i])

# print(np.size(evfit[0,11]))
# print(coord[11])
# #plt.plot(ev[coord[11]:coord[11]+np.size(evfit[0,11])])
# plt.plot(ev[int(coord[11]):int(coord[11])+218])
# plt.show()
```

```
## TIME ##
time4 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    time4[i] = np.size(evfit[0,i])

## HEIGHT ##
h4 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    #h[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h4[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i])
## MEAN ##
mu4 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    mu4[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(ev

# mu4 = (mu4 - mu4.mean()) / mu4.std()
# time4 = (time4 - time4.mean()) / time4.std()
# h4 = (h4 - h4.mean()) / h4.std()
## NLEVELS ##
lvl4 = mat['EventDatabase']
lvl4 = lvl4['NumberOfLevels']
lvls4 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    lvls4[i] = np.asscalar(lvl4[0,i])

############

os.chdir("/Users/ANGEL/Desktop/HiWi/Data Aleksandra/Homopolymers"

mat = scipy.io.loadmat('A2.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(np.size(crd))
fits = mat['ConcatenatedFits']
evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(np.size(crd))

ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(np.size(crd)):
    coord[i] = np.asscalar(crd[0,i])
    #evfitt[i] = np.asscalar(evfit[0,i])
```

```
# print(np.size(evfit[0,11]))
# print(coord[11])
# #plt.plot(ev[coord[11]:coord[11]+np.size(evfit[0,11])])
# plt.plot(ev[coord[11]+173+17:coord[11]+173+17+18])
# plt.show()


## TIME ##
time5 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    time5[i] = np.size(evfit[0,i])

## HEIGHT ##
h5 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    #h[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h5[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i])
## MEAN ##
mu5 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    mu5[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(ev

# mu4 = (mu4 - mu4.mean()) / mu4.std()
# time4 = (time4 - time4.mean()) / time4.std()
# h4 = (h4 - h4.mean()) / h4.std()
## NLEVELS ##
lvl5 = mat['EventDatabase']
lvl5 = lvl5['NumberOfLevels']
lvls5 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    lvls5[i] = np.asscalar(lvl5[0,i])

#########

os.chdir("/Users/ANGEL/Desktop/HiWi/Data Aleksandra/Homopolymers"

mat = scipy.io.loadmat('A3.mat')

crd = mat['EventDatabase']
crd = crd['ConcatenatedStartCoordinates']
coord = np.ndarray(np.size(crd))
fits = mat['ConcatenatedFits']
evfit = mat['EventDatabase']
evfit = evfit['AllLevelFits']
evfitt = np.ndarray(np.size(crd))
```

```
ev = mat['ConcatenatedEvents']
fits = mat['ConcatenatedFits']

for i in range(np.size(crd)):
    coord[i] = np.asscalar(crd[0,i])
    #evfitt[i] = np.asscalar(evfit[0,i])

# print(np.size(evfit[0,11]))
# print(coord[11])
# #plt.plot(ev[coord[11]:coord[11]+np.size(evfit[0,11])])
# plt.plot(ev[coord[11]+173+17:coord[11]+173+17+18])
# plt.show()


## TIME ##
time6 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    time6[i] = np.size(evfit[0,i])

## HEIGHT ##
h6 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    #h[i] = max(ev[coord[i]:coord[i]+np.size(evfit[0,i])])
    h6[i] = max(ev[int(coord[i]):int(coord[i]+np.size(evfit[0,i]))
## MEAN ##
mu6 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    mu6[i] = np.mean((fits[int(coord[i]):int(coord[i]+(np.size(ev

# mu4 = (mu4 - mu4.mean()) / mu4.std()
# time4 = (time4 - time4.mean()) / time4.std()
# h4 = (h4 - h4.mean()) / h4.std()
## NLEVELS ##
lvl6 = mat['EventDatabase']
lvl6 = lvl6['NumberOfLevels']
lvls6 = np.ndarray(np.size(crd))
for i in range(np.size(crd)):
    lvls6[i] = np.asscalar(lvl6[0,i])


#####
# # analyze 1 level #
index=np.where(lvls==1)
index2=np.where(lvls2==1)
index3=np.where(lvls3==1)
```

```
index4=np.where(lvls4!=1)
# time = (time - time.mean())/ time.std()
# time2 = (time2 - time2.mean())/ time2.std()
# time3 = (time3 - time3.mean())/ time3.std()
# mu = (mu - mu.mean()) / mu.std()
# mu2 = (mu2 - mu2.mean()) / mu2.std()
# mu3 = (mu3 - mu3.mean()) / mu3.std()
# h3 = (h3 - h3.mean()) #/ h3.std()
# ####

# h4 = (h4 - h4.mean()) / h4.std()
# h5 = (h5 - h5.mean()) / h5.std()
# h6 = (h6 - h6.mean()) / h6.std()
# time4 = (time4 - time4.mean())/ time4.std()
# time5 = (time5 - time5.mean())/ time5.std()
# time6 = (time6 - time6.mean())/ time6.std()
# mu4 = (mu4 - mu4.mean()) / mu4.std()
# mu5 = (mu5 - mu5.mean()) / mu5.std()
# mu6 = (mu6 - mu6.mean()) / mu6.std()

# index=np.where(lvls!=1)
# index2=np.where(lvls2!=1)
# index3=np.where(lvls3!=1)
# data = np.ndarray(shape=(48,2))
# #####
# data[0:26,0] = time[index]
# data[26:48,0] = time2[index2]
# # data[28:33,0] = time3[index3]
# data[0:26,1] = -mu[index]
# data[26:48,1] = -mu2[index2]
# #data[28:33,1] = -mu3[index3]
############
# index=np.where(lvls==1)
# index2=np.where(lvls2==1)
# index3=np.where(lvls3==1)
# data = np.ndarray(shape=(28,2))
# ##
# data[0:19,0] = time[index]
# data[19:28,0] = time2[index2]
# # data[28:33,0] = time3[index3]
# data[0:19,1] = -mu[index]
# data[19:28,1] = -mu2[index2]
# #data[28:33,1] = -mu3[index3]
###################
data = np.ndarray(shape=(3887,2))
data[0:1660,0] = -mu4
```

```
data[1660:1688,0] = -mu5
data[1688:3887,0] = -mu6
#data[89:130,0] = time4
data[0:1660,1] = time4
data[1660:1688,1] = time5
data[1688:3887,1] = time6
#data[89:130,1] = -mu4
# data[0:1660,2] = lvls4
# data[1660:1688,2] = lvls5
# data[1688:3887,2] = lvls6
#data[89:130,2] = h4
###################
# plt.plot(data[:,0],data[:,1],'+',markersize=2)
# plt.xlabel("Dwell time ($\mu$s)", fontsize=15);
# plt.ylabel("Mean (blue) / Height (orange) (nA)", fontsize=15);
# # plt.xlabel("Mean (nA)", fontsize=15);
# # plt.ylabel("Height (nA)", fontsize=15);
# plt.show()
#
# plt.plot(time4,-mu4,'+')
# #plt.plot(time4,-mu4,'+')
# plt.xlabel("Dwell time ($\mu$s)", fontsize=12);
# plt.ylabel("Mean (blue) / Height (orange) (nA)", fontsize=12);
# plt.show()

# data = np.ndarray(shape=(np.size(crd),3))
# data[:,0] = h4
# data[:,1] = -mu4
# data[:,2] = time4
# ## 3dplot ##
# ax = plt.axes(projection='3d')
# ax.scatter(data[:,0], data[:,1], data[:,2], c=data[:,2], cmap='
# #ax.scatter(time4, -mu4, h4, c=h4)
# # ax.set_xlabel('Number of levels',fontsize=9)
# # ax.set_ylabel('Dwell time',fontsize=9)
# # ax.set_zlabel('Height', fontsize=9)
# plt.show()

# data = np.ndarray(shape=(1100,2))
# data[:,0] = h6
# data[:,1] = -mu6[0:1100]
#
# data[:,2] = mu
# db = DBSCAN(eps=0.2).fit(data)
# core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
# core_samples_mask[db.core_sample_indices_] = True
# labels = db.labels_
# # Number of clusters in labels, ignoring noise if present.
# n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
#
# print('Estimated number of clusters: %d' % n_clusters_)


silou = np.ndarray(11)
### silhoutte score ###
chscore = np.ndarray(11)
# data, y = make_blobs(n_samples=500,
#                      n_features=2,
#                      centers=4,
#                      cluster_std=1,
#                      center_box=(-10.0, 10.0),
#                      shuffle=True,
#                      random_state=1)  # For reproducibility

range_n_clusters = [2,3,4,5,6,7,8,9,10]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 8)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(data) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=3)
    cluster_labels = clusterer.fit_predict(data)

    # The silhouette_score gives the average value for all the sa
    # This gives a perspective into the density and separation of
    # clusters
    silhouette_avg = silhouette_score(data, cluster_labels)
    silou[n_clusters] = silhouette_avg
    chscore[n_clusters] = metrics.calinski_harabaz_score(data, cl
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg,
          "CH Score", chscore[n_clusters])

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(data, cluster_la

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=

        # Label the silhouette plots with their cluster numbers at
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the va
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.spectral(cluster_labels.astype(float) / n_clusters
    ax2.scatter(data[:, 0], data[:, 1], marker='.',  s=50, lw=0, a
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
```

```
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=900, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=120, edgecolor='k')

    #ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel(("Height (nA)"),fontsize=17)
    ax2.set_ylabel(("Mean (nA)"),fontsize=17)
    ax2.tick_params(axis='both', which='major', labelsize=15)
    # plt.suptitle(("Silhouette analysis for KMeans clustering on
    #                "with n_clusters = %d" % n_clusters),
    #                fontsize=15, fontweight='bold')
    extent = ax2.get_window_extent().transformed(fig.dpi_scale_tra
    #fig.savefig('ax2_figure.png', bbox_inches=extent)
    # Pad the saved area by 10% in the x-direction and 20% in the
    fig.savefig('ax2A_figure_expanded.png', bbox_inches=extent.exp
    plt.show()

#chscore[2:11] = (chscore[2:11] - max(chscore))/(max(chscore)- mi
#############################
normalized = (chscore[2:11]-min(chscore[2:11]))/(max(chscore[2:11
results = np.ndarray(shape=(3887,3))
results[:,0] = data[:,0]
results[:,1] = data[:,1]
results[:,2] = cluster_labels
#scipy.io.savemat('/Users/ANGEL/Desktop/MLarticle papers/FIGURAS/
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar  5 17:37:50 2019

@author: angel
"""

#import tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import numpy as np
from keras.utils import np_utils
import pandas as pd
import scipy.io
import matplotlib.pyplot as plt
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

def build_regressor():
    regressor = Sequential()
    regressor.add(Dense(units=3, input_dim=3, activation='relu'))
    regressor.add(Dense(5, activation='relu'))
    regressor.add(Dense(5, activation='relu'))
    regressor.add(Dense(5, activation='relu'))
    regressor.add(Dense(units=1))
    regressor.compile(optimizer='adam', loss='mean_squared_error',
    regressor.evaluate(test_x, test_y)
    return regressor

#Import training dataset
#training_dataset = pd.read_csv('iris_training.csv', names=COLUMN_
#train_x = training_dataset.iloc[:, 0:4].values
#train_y = training_dataset.iloc[:, 4].values
training_dataset = scipy.io.loadmat('AdBHeight2.mat')
sorted(training_dataset.keys())
train_x = training_dataset['adforH']
train_x = train_x[0:3000,1:4]
train_y = training_dataset['adforH']
train_y = train_y[0:3000,1]
## Import testing dataset
#test_dataset = pd.read_csv('iris_test.csv', names=COLUMN_NAMES,
test_x = training_dataset['adforH']
test_x = test_x[3000:3886,1:4]
test_y = training_dataset['adforH']
test_y = test_y[3000:3886,1]

#
## Encoding training dataset
##encoding_train_y = np_utils.to_categorical(train_y)
#
## Encoding training dataset
##encoding_test_y = np_utils.to_categorical(test_y)
#
# Creating a model
#model = Sequential()
#model.add(Dense(3, input_dim=3, activation='relu'))
#model.add(Dense(3, activation='relu'))
#model.add(Dense(1, activation='softmax'))

# Compiling model
#model.compile(loss='mean_squared_error', optimizer='adam', metri

# Training a model
regressor = KerasRegressor(build_fn=build_regressor, batch_size=1
results=regressor.fit(train_x,train_y)
#model.fit(train_x, train_y, epochs=10, batch_size=10)
seed =7
y_pred= regressor.predict(test_x)


# Evaluate the model

train_error = np.abs(test_y - y_pred)
train_error2 =  np.sum(np.abs(test_y - y_pred))/len(test_y)
mean_error = 100 - np.mean(train_error)
min_error = np.min(train_error)
max_error = np.max(train_error)
std_error = np.std(train_error)
print(mean_error)
#print(min_error)
#print(max_error)
#print(std_error)

#print("\nAccuracy: %.2f%%" % (scores[1]))

fig, ax = plt.subplots()
ax.scatter(test_y, y_pred)
ax.plot([test_y.min(), test_y.max()], [test_y.min(), test_y.max()
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()

results = np.ndarray(shape=(886,2))
results[:,0] = test_y
results[:,1] = y_pred
scipy.io.savemat('DTpred.mat',mdict={'DTpredd':results})
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Mar  5 17:37:50 2019

@author: angel
"""

#import tensorflow
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Embedding
from keras.layers import Convolution1D, GlobalAveragePooling1D, Ma
import numpy as np
import scipy.io
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import numpy as np
from keras.utils import np_utils
import pandas as pd
import scipy.io
import matplotlib.pyplot as plt
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from keras.optimizers import SGD
import random
from IPython.display import clear_output


class PlotLosses(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []

        self.fig = plt.figure()

        self.logs = []

    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.i += 1

        clear_output(wait=True)
        plt.plot(self.x, self.losses, label="loss")
        plt.plot(self.x, self.val_losses, label="val_loss")
        plt.title('model loss')
        plt.legend()
        plt.show();

plot_losses = PlotLosses()



nb_class = 3
nb_features = 4

training_dataset1 = scipy.io.loadmat('AdBfeat.mat')
sorted(training_dataset1.keys())
training_dataset2 = scipy.io.loadmat('GuAfeat.mat')
sorted(training_dataset2.keys())
training_dataset3 = scipy.io.loadmat('CyBfeat.mat')
sorted(training_dataset3.keys())

train_x1 = training_dataset1['ad']
train_x1 = train_x1[0:500,0:4]
train_x2 = training_dataset2['GuAfeat']
train_x2 = train_x2[0:500,0:4]
train_x3 = training_dataset3['cyBfeat']
train_x3 = train_x3[150:650,0:4]

data_x = np.concatenate((train_x1, train_x2, train_x3), axis=0)
train_x = data_x

test_x1 = training_dataset1['ad']
test_x1 = test_x1[2000:2150,0:4]
test_x2 = training_dataset2['GuAfeat']
test_x2 = test_x2[500:650,0:4]
test_x3 = training_dataset3['cyBfeat']
test_x3 = test_x3[0:150,0:4]
datat_x = np.concatenate((test_x1, test_x2, test_x3), axis=0)

test_x = datat_x
```

```
test_datasetxd3 = scipy.io.loadmat('XD3feat.mat')
test_xd3 = test_datasetxd3['xd3']
test_xd3 = test_xd3[0:390,0:4]

###
## 1 = Ad, 2 = Gu, 3 = Cy #####
train_y1 = np.full((1500,1), 1)
train_y1 = train_y1[0:500]
train_y2 = np.full((650,1), 2)
train_y2 = train_y2[0:500]
train_y3 = np.full((650,1), 3)
train_y3 = train_y3[0:500]

data_y = np.concatenate((train_y1, train_y2, train_y3), axis=0)
train_y = data_y

test_y1 = train_y1[500:650]
test_y2 = train_y2[500:650]
test_y3 = train_y3[500:650]

test_y1 = np.full((150,1), 1)
test_y2 = np.full((150,1), 2)
test_y3 = np.full((150,1), 3)

datat_y = np.concatenate((test_y1, test_y2, test_y3), axis=0)
test_y = datat_y

#train_y = np_utils.to_categorical(train_y, nb_class)
#test_y = np_utils.to_categorical(test_y, nb_class)

# ================================================================
# # reshape train data
# train_x2 = np.zeros((len(train_x), nb_features, 3))
# train_x2[:, :, 0] = train_x[:, :nb_features]
# train_x2[:, :, 1] = train_x[:, nb_features:128]
# train_x2[:, :, 2] = train_x[:, 128:]
#
# # reshape validation data
# test_x2 = np.zeros((len(test_x), nb_features, 3))
# test_x2[:, :, 0] = test_x[:, :nb_features]
# test_x2[:, :, 1] = test_x[:, nb_features:128]
# test_x2[:, :, 2] = test_x[:, 128:]
# ================================================================
from keras.models import Sequential
from keras.layers import Convolution2D, Dense, Dropout, Flatten, M
from keras.utils import np_utils
import numpy as np

# import your data here instead
# X - inputs, 10000 samples of 128-dimensional vectors
# y - labels, 10000 samples of scalars from the set {0, 1, 2}


# process the data to fit in a keras CNN properly
# input data needs to be (N, C, X, Y) - shaped where
# N - number of samples
# C - number of channels per sample
# (X, Y) - sample size

train_x = train_x.reshape((1500, 1, 4, 1))
test_x11 = test_x
test_x = test_x.reshape((450, 1, 4, 1))

test_xd3 = test_xd3.reshape((390, 1, 4, 1))
# output labels should be one-hot vectors - ie,
# 0 -> [0, 0, 1]
# 1 -> [0, 1, 0]
# 2 -> [1, 0, 0]
# this operation changes the shape of y from (10000,1) to (10000,

train_y = np_utils.to_categorical(train_y)
test_y11 = test_y
test_y = np_utils.to_categorical(test_y)
# define a CNN
# see http://keras.io for API reference
# import regularizer
from keras.regularizers import l2
# instantiate regularizer
reg = l2(0.001)



cnn = Sequential()
cnn.add(Convolution2D(30, 3, 1,
    border_mode="same",
    activation="relu", activity_regularizer=reg,
    input_shape=(1, 4, 1)))
cnn.add(Convolution2D(30, 3, 1, border_mode="same", activation="re
cnn.add(MaxPooling2D(pool_size=(1,1)))
#
#cnn.add(Convolution2D(128, 3, 1, border_mode="same", activation=
#cnn.add(Convolution2D(128, 3, 1, border_mode="same", activation=
#cnn.add(Convolution2D(128, 3, 1, border_mode="same", activation=
#cnn.add(MaxPooling2D(pool_size=(1,1)))
#
#cnn.add(Convolution2D(256, 3, 1, border_mode="same", activation=
#cnn.add(Convolution2D(256, 3, 1, border_mode="same", activation=
#cnn.add(Convolution2D(256, 3, 1, border_mode="same", activation=
#cnn.add(MaxPooling2D(pool_size=(1,1)))

cnn.add(Flatten())
cnn.add(Dense(1024, activation="relu", activity_regularizer=reg))
#cnn.add(Dropout(0.5))
cnn.add(Dense(4, activation="softmax"))

# define optimizer and objective, compile cnn

cnn.compile(loss="categorical_crossentropy", optimizer="adam",metr

# train

history = cnn.fit(train_x, train_y, epochs=100, #batch_size=50,
                  shuffle=True, callbacks=[plot_losses],
                  validation_data=(test_x,test_y))

y_pred = cnn.predict_classes(test_x)
y_pred2 = np_utils.to_categorical(y_pred)
#y_pred = np_utils.to_categorical(y_pred)
score, acc = cnn.evaluate(test_x, test_y, verbose=0)
print('Test loss:', score)
print('Test accuracy:', acc)


plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
#train_error = np.abs(test_y - y_pred)
#train_error2 =  np.sum(np.abs(test_y - y_pred))/len(test_y)
#mean_error = 100 - np.mean(train_error)
```

```
results = np.ndarray(shape=(100,2))
results[:,0] = history.history['acc']
results[:,1] = history.history['val_acc']
scipy.io.savemat('DataAcc.mat',mdict={'DataAccc':results})
```