Predicting Antibody Developability: Machine Learning Meets Therapeutic Antibodies

William Björkhem, Josephine Höjding

Master's thesis
2025:E33

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Predicting Antibody Developability: Machine Learning Meets Therapeutic Antibodies

Written by: Josephine Höjding & William Björkhem
Examiner: Kalle Åström (Lund University)
Main Supervisor: Mikael Nilsson (Lund University)
Co-supervisor: Morten Krogh (Bionamic AB)

Abstract

Antibody developability refers to an antibody's suitability for clinical use, including properties such as solubility, stability, and aggregation. These traits are traditionally assessed through experimental screening, which is time-consuming and resource heavy. Machine learning offers a promising alternative for early prediction of developability, though many existing models are still in early stages.

This work compares multiple machine learning strategies for predicting protein solubility, a key developability factor. Five datasets were used: four consisting of non-antibody protein sequences expressed in E. Coli with solubility labels, and one independent antibody dataset without labels. Three existing models—NetSolP, SWI, and ProteinSol—were evaluated using standard performance metrics, and new models were developed by leveraging feature extraction from SWI and ProteinSol to explore potential improvements.

Developed approaches included logistic regression for direct solubility prediction, models that first classified a sample's likely dataset of origin before applying a corresponding solubility model, clustering-based methods with cluster-specific classifiers, and multi-layer perceptrons to test the benefits of deeper architectures.

Overall, the models achieved similar performance, with no single approach consistently outperforming others. Simpler models like logistic regression often performed on par with more complex models such as multi-layer perceptrons. Results varied by dataset, with the lowest performance observed on the largest and most diverse dataset, PDBSol, suggesting that high variability in sequence data may reduce prediction reliability.

Acknowledgements

Firstly, we would like to thank our supervisors, Mikael Nilsson at LTH and Morten Krogh at Bionamic AB, for the helpful discussions and guidance during our master thesis work. We have appreciated your valuable feedback and insights to our project.

We would also like to thank our colleagues at Bionamic AB for their support and encouragement, and for providing a welcoming work environment.

Josephine Höjding & William Björkhem
Lund, Sweden, 2025

Contents

CHAPTER 1

Introduction

1.1 Background

The first ever therapeutic antibody approved for clinical use was Muromonab-CD3, designed to prevent transplant rejection [1]. Approved by the United States Food and Drug Administration (US FDA) in 1986, this marked the beginning of a rapidly expanding global market for these protein complexes. As of early 2025, a total of 229 therapeutic antibodies have been approved or are in review worldwide [2]. Since Muromonab-CD3's approval in 1986, there has been a notable increase in the development and approval of laboratory-created antibodies, called monoclonal antibodies (mAbs), as shown in Figure 1.1 [3].

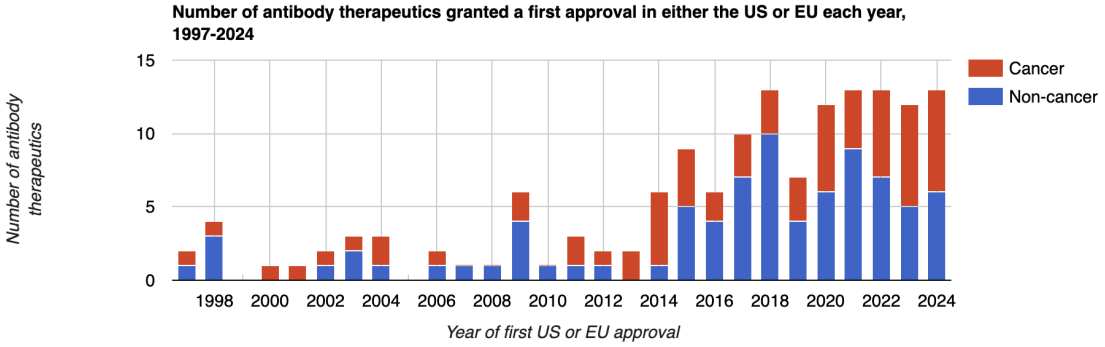


FIGURE 1.1: The number of therapeutic antibodies granted a first approval in the US or EU each year from 1997-2024 [3]. The red bar corresponds to therapeutics developed to treat various cancers and the blue to non-cancer diseases.

Therapeutic antibodies have demonstrated efficacy in a wide range of diseases with oncology, immunology, and hematology being the dominant fields [1]. Their increasing popularity over the years is attributed to their high specificity and minimal adverse effects when treating diseases. Furthermore, antibody engineering has experienced significant advancements during the years following the creation of Muromonab-CD3 with mAbs becoming the predominant class of newly produced drugs. A selection of key therapeutic antibodies, their initial treatment area, and years of approval are provided in Figure 1.2, showing the wide range of treatment areas.

mAb	**Initial Treatment Area**	**US Approval Year**
Muromonab-CD3	Kidney transplant rejection	1986
Rituximab	Non-Hodgkin lymphoma	1997
Trastuzumab	Breast cancer	1998
Infliximab	Crohn's disease	1998
Adalimumab	Rheumatoid arthritis	2002
Pembrolizumab	Melanoma	2014
Nivolumab	Melanoma, non-small cell lung cancer	2014
Casirivimab and imdevimab	COVID-19	2020

FIGURE 1.2: A selection of important mAbs, their initial treatment area, and what year they were approved by the US FDA. The antibodies are compiled from [1] and [2].

The rapid expansion of therapeutic antibody development has driven substantial growth in the global mAb market [1] and projections suggest continued expansion over the upcoming years [4, 5]. While most sources agree that the market will grow significantly, projections for 2034 vary widely from 581.42 billion USD [4] to 973.60 billion USD [5]. Figure 1.3 illustrates the historical and predicted market value over the years up until year 2034.

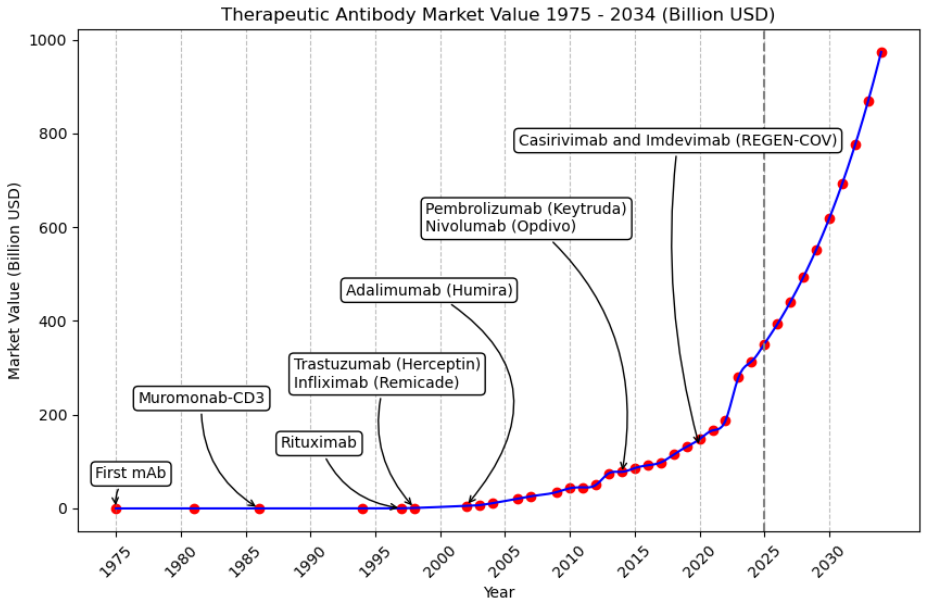


FIGURE 1.3: The global market value growth of therapeutic antibodies. The figure is inspired by a similar figure in [1]. The market value between years 2019-2022 were taken from [4], and 2023-2034 from [5]. A dashed line marks the current year (2025) and the values from this point onward are predictions. A selection of key mAbs with their corresponding brand names are also plotted in their corresponding years of approval.

For an antibody to be viable as a therapeutic candidate, it must possess certain favorable traits, such as high stability and solubility [6]. Solubility refers to the ability of an antibody to remain in solution at relevant concentrations without forming aggregates, which is crucial for ensuring proper formulation, delivery and long-term stability [7]. Low solubility can lead to increased viscosity, self-association, and aggregation, all of which can compromise the efficacy, safety, and manufacturability of the therapeutic [6, 7]. Traditionally, these properties have been evaluated through experimental screening, a process that is time-consuming and costly [6, 8]. This challenge has opened up opportunities for machine learning algorithms to streamline the process.

Machine learning models provide an alternative way to predict antibody characteristics such as aggregation risk, solubility, and potential immune responses in early stages of clinical trials [6]. This allows for elimination of poor candidates early in the pipeline before expensive lab testing of the antibodies. Advanced deep learning models, such as large language models used to learn underlying patterns from antibody sequence data, have already been developed for this cause [9]. These machine learning models provide high-throughput screening, narrowing the candidate pool down to fewer and more promising mAbs instead of relying on experimental testing of millions of antibodies.

The emergence of artificial intelligence (AI) and machine learning models in the biomedical

field has become evident as demonstrated by the most recent Nobel Prize in Chemistry partly being awarded to the developers of the AI system AlphaFold2 [10]. This tool, developed by Google DeepMind, solved the complex problem of predicting protein structures with high accuracy. This area was previously dominated by costly and time-consuming methods, such as X-ray crystallography. The usage of AlphaFold2 allows for a similar accuracy in three-dimensional protein structure prediction as conventional methods but provides the prediction in a matter of minutes and is less resource heavy. The highly acknowledged AI system can be considered a clear sign of how artificial intelligence is evolving in the biomedical field and gaining importance in healthcare research.

1.2 Aims and Objectives

The increasing reliance on therapeutic antibodies in medicine, coupled with the rapid expansion of the global market, underscores the need for more efficient and cost-effective developability assessment methods. Traditional experimental screening remains resource-intensive, highlighting the potential of AI models to streamline the clinical trial process of viable antibody candidates. Despite advancements in AI-driven approaches, there is still a need for further evaluation of different prediction models for antibody developability to identify promising candidates for practical application.

This work evaluates various machine learning models for predicting solubility, a key antibody characteristic. By analyzing their predictive performance, the study seeks to identify models that could serve as strong candidates for integration in the antibody development pipeline. Additionally, this work explores opportunities to create and further refine AI-based methods, contributing to the ongoing advancement of computational approaches in antibody engineering.

To achieve this, the following objectives are defined:

- Evaluate the predictive capabilities of different existing machine learning algorithms in predicting antibody solubility.
- Design, implement, and optimize AI-based models to improve predictive accuracy and reliability for antibody solubility assessment.
- Analyze model performance and feature contributions to identify informative or redundant features across datasets and tools.
- Determine which machine learning models provide the most accurate and robust predictions across diverse evaluation metrics using standardized datasets.

1.3 Limitations

While this study aims to evaluate and compare machine learning models for antibody developability, several limitations must be acknowledged.

- **Limited Model Selection:** Only a subset of models was evaluated due to accessibility challenges. Even though many models are open to the public with open-source code, they can often be incompatible with the system environment due to factors such as deprecated libraries, operating system constraints, or lack of proper documentation.
- **Focus on Solubility Prediction:** All models used and developed in this study were designed specifically to predict solubility, one aspect of antibody developability. This choice was made because developability includes multiple interrelated characteristics, and comparing models trained on different traits would have introduced additional complexity and inconsistency.
- **Dataset Constraints:** The availability of high quality datasets for training and evaluation were limited. Many existing datasets lacked comprehensive ground truth labels, and in cases where labeled data was available, its accuracy was sometimes uncertain due to the lack of clarity in the labeling process and the potential use of several methods to determine solubility. Additionally, most of the datasets used in this study included non-antibody proteins expressed in E. coli, not therapeutic antibodies, because well-labeled antibody solubility data was difficult to obtain. This mismatch may have limited the ability for model generalization of real antibody developability tasks.
- **Sequence Data Limitation:** All models evaluated in this study relied on antibody sequence data. While there are models that integrate structural data for potentially more accurate predictions, obtaining such data can be challenging. Structural data often require pre-existing structure files or prediction using computationally intensive methods like AlphaFold2, which may not always be feasible with resource contraints.

These limitations highlight the challenges in machine learning-driven antibody developability prediction and suggest areas for future research, such as broader model evaluations, multi-trait developability predictions, and improved dataset curation.

CHAPTER 2

Theory

2.1 Antibodies

Antibodies, also called immunoglobulins, are specialized proteins that identify and neutralize foreign substances such as bacteria, viruses, and toxins [11]. They are produced by the immune system and are essential in the body's defense mechanism. They work by binding to pathogens or antigens, which are unique molecules typically found on the surface of—or secreted by—the invaders. Antibodies have several functions, including neutralization, opsonization, and complement activation. Neutralization involves directly binding to harmful pathogens and preventing them from entering host cells and causing infection. Opsonization entails coating the pathogens and thereby marking them to be destroyed by other immune cells. Lastly, they can also trigger a cascade of immune responses with complement activation.

Antibodies have a Y-shaped structure (Figure 2.1) and are composed of two heavy and two light protein chains with disulfide bonds in between [11]. The light chains are located in the arms of the Y-shaped structure, where they pair with the heavy chains while the heavy chains span through the entire structure. Antibodies have a variable section at the ends of the arms which determines the antigen specificity while the rest of the structure remains constant.

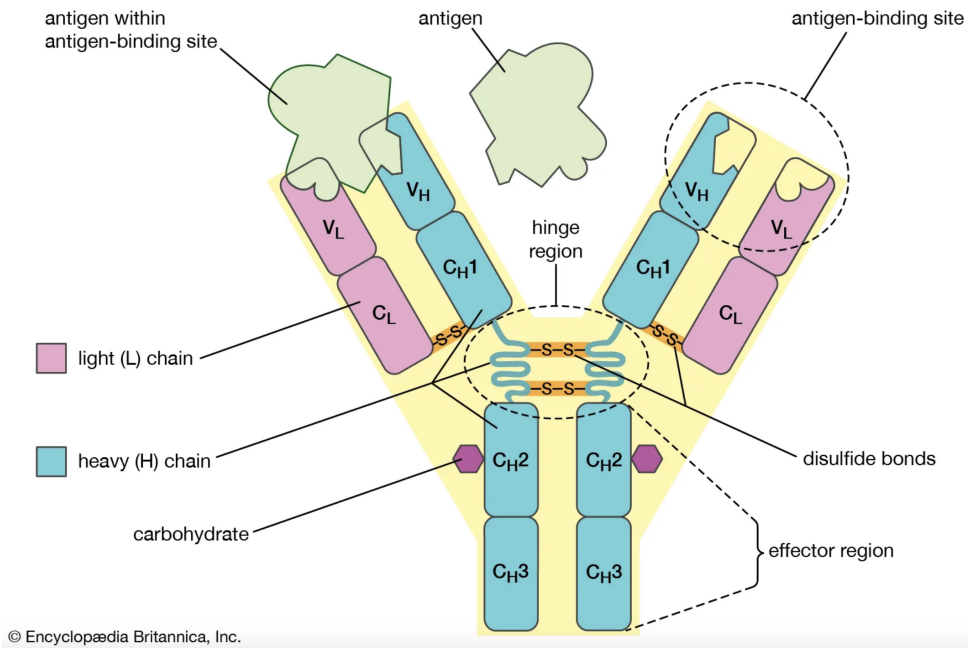


FIGURE 2.1: The structure of an antibody [11]. C and V indicate constant and variable regions respectively.

2.1.1 Proteins

Proteins are molecules that play a critical role in several biological processes [12]. They are composed of long chains of amino acids, which are organic compounds containing an amino group (-NH_2) and a carboxyl group (-COOH) attached to a central carbon atom. Proteins found naturally consist of around 20 different types of amino acids.

As seen in Figure 2.2, the structure of a protein can be divided into four hierarchical levels [12]. The primary structure refers to the linear sequence of amino acids, linked by peptide bonds. The order of amino acids determines the protein's characteristics. The secondary structure alludes to the folding patterns of the protein which are stabilized by hydrogen bonds. This structure includes configurations called α-helices and β-sheets. The tertiary structure describes the three-dimensional shape of the protein due to interactions between side chains of the protein. A quaternary structure can be found when a protein consists of several amino acid sequences called subunits. The structure refers to the arrangement of these subunits into a complex.

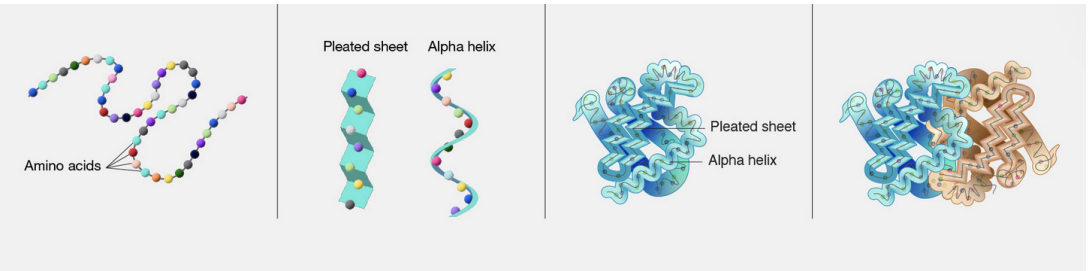


FIGURE 2.2: From left to right: the primary, secondary, tertiary, and quaternary structure of a protein [13]. Pleated sheets are often called β-sheets.

2.1.2 Antibodies in Medicine

Antibodies' ability to recognize and bind to antigens make them valuable tools in medical applications [11]. They are naturally produced in response to vaccines, helping the body to prepare for future infections. They are also used in medical tests to detect infections and conditions, and can be used to help reduce allergic reactions. An area where antibodies are widely used is in the form of monoclonal antibodies (mAbs), which are laboratory-made antibodies created to target specific diseases. The mAbs have dominated in recent pharmaceutical developments due to their abilities to have high specificity (see Section 2.2.2) and minimal adverse effects [14].

Therapeutic antibodies are used to treat several diseases such as cancer, autoimmune diseases, and infectious diseases [11]. They can help target cancer cells more precisely than chemotherapy which reduces side effects and increases efficacy [15]. It is common to use a combination of mAbs and chemotherapy to treat cancer as well as using antibodies as immunoconjugates to deliver cytotoxic, i.e., toxic to cells, substances directly to cancer cells. Antibodies can also be used to treat autoimmune diseases by targeting overactive immune systems and reducing inflammation. Arthritis is an example of a disease where mAbs can be used to reduce symptoms such as joint damage. In infectious diseases, mAbs can be designed to target viral proteins which inhibits the virus from entering cells [16]. It is also possible for antibodies to neutralize bacterial infections by targeting toxins released by bacteria, removing the potential harm they can cause. For high risk groups, mAbs have been used to try to prevent sickness in diseases such as respiratory syncytial virus (RSV), Ebola, and COVID-19. This enables potential prevention of diseases for individuals who might not be able to produce an effective immune response on their own.

2.2 Antibody Developability

Antibody developability refers to the ability of an antibody to be successfully developed into a therapeutic drug [6]. In terms of early-stage discovery, it involves assessing an antibody's ability to be able to go from a laboratory concept into a market-ready drug. This consists of

evaluating certain characteristics such as stability, solubility, manufacturability, immunogenicity, and overall performance in an in vivo environment. Integrating developability assessments into the early-stage phase can increase the likelihood of successful drug candidates and minimize risk of failure later in development.

2.2.1 Physicochemical Properties

Physicochemical properties such as those mentioned below can influence the developability of antibodies in many ways and are crucial factors in the evaluation of antibodies in their development.

Solubility

In certain uses of therapeutic antibodies, such as in ophthalmology, they are injected intravitreally in small doses, requiring antibodies to be highly soluble [6]. An antibody's solubility is affected by its self-interaction inversely. This means that increased interactions result in lower solubility. These interactions are driven by van der Waals and hydration forces as well as hydrophobic (see Section 2.2.1) and electrostatic (see Section 2.2.1) interactions, all closely linked to structural properties. Antibodies with poor solubility are more susceptible to aggregation (see Section 2.2.1) which can lead to issues involving off-target binding and rapid clearance from the body. Soluble antibodies are easier to produce and purify during large-scale manufacturing, as they are less likely to aggregate or form insoluble clumps compared to poorly soluble antibodies, which can cause complications [17].

Stability

In developing therapeutic antibodies, stability affects formulation, storage, and efficacy [6][17]. Poor stability is not ideal since it can lead to antibody aggregation (see Section 2.2.1), loss of activity, immunogenicity (see Section 2.2.3), or toxicity.

Thermal stability is essential in order to maintain an antibody's structure and efficacy when exposed to temperature changes in for example storage and shipping [6]. High thermal stability is positively correlated with a greater number of hydrogen bonds in the molecule as well as the fraction of an antibody's surface area that is polar. In manufacturing and storage, antibodies are often exposed to light which can cause oxidation of light sensitive amino acids. Establishing safe lighting during these processes, ensures that the antibody solution is light-stable and avoids color changes and aggregation (see Section 2.2.1). Stability is also crucial when it comes to pH-sensitivity. During manufacturing, antibodies can be exposed to low pH environments which

can cause breaking of the protein structure and increased risk of aggregation. This can similarly happen with applied physical stress in for example transport and administration. Surfactants are often used in this case to reduce the risk of physical instability.

Aggregation

Aggregation refers to the process where individual antibody molecules associate together, forming larger complexes or aggregates [6]. These are more likely to promote immunogenicity (see Section 2.2.3), decrease the biological function of the antibody by interfering with antigen-binding (see Section 2.2.2), and increase viscosity. Aggregation is influenced by many factors such as folding stability, surface charge distribution (see Section 2.2.1), and hydrophobicity (see Section 2.2.1) among others. Aggregated forms are difficult to purify due to their tendency to bind non-specifically to purification matrices, resulting in lower production efficiency and higher cost when manufacturing. It can also cause a shorter shelf life due to aggregation often indicating instability of the antibody.

Structure

The structure of a protein can largely affect how soluble and developable a protein is [18]. Interactions between hydrophobic patches of the protein can occur if improper folding arises which in turn can lead to aggregation. There is also a higher likelihood of degradation and loss of function with misfolded proteins which hinder clinical efficacy and safety [19]. A misfolded protein increases immunogenicity (see Section 2.2.3) as it is recognized as a foreign object in the body. Structural integrity is also vital for maintaining high binding affinity and binding specificity (see Section 2.2.2). This is due to the structure directly contributing to the unique binding sites of the antibody which are used to recognize and bind to antigens.

Protein structure flexibility refers to a protein's ability to undergo conformational changes without losing structural integrity [20]. A method to measure flexibility is to derive B-factors using X-ray crystallography which indicate atomic displacement withing a protein structure. Higher values of B-factors indicate greater atomic flexibility while lower values points to increased rigidness. Proteins with higher flexibility may have better solubility characteristics due to the reduced aggregation tendencies [21]. However, if the protein is excessively flexible, there is a higher risk for misfolding and instability.

Amino Acid Composition

The amino acid composition refers the percentage of each amino acid within the protein sequence. This can have an impact on an antibody's developability due to the characteristics of single amino acids [6]. Certain amino acids have the ability to undergo processes where their functional groups are removed or converted into other functional groups which could reduce the antibody's potency. The amino acids can also have different levels of hydrophobicity (see Section 2.2.1). For certain regions on the antibody, hydrophobic residues can increase the protein's tendency to aggregate.

Charge Distribution

The charge distribution across the surface of a protein or antibody can affect its solubility [6]. When an antibody has regions with highly concentrated positive or negative charges, it can increase self-interactions. These can cause aggregation and therefore decrease solubility. The electrostatic repulsion between parts of the antibody can be maintained with more evenly distributed charges and will therefore be favorable for development.

Length

The length of proteins is simply the number of amino acids in the sequence. This can influence the physical and chemical properties of antibodies. Protein length has been show to correlate with other properties such as stability [22], isoelectric point [23] (see Section 2.2.1), and deleterious mutations [24].

Isoelectric Point (pI)

The isoelectric point (pI) refers to the pH at which a molecule, such as a protein, has no net electrical charge [25]. At this pH, the number of positive and negative charges within the molecule are balanced. In proteins, the pI is largely determined by the ionizable groups present in the amino acids side chains and at certain ends of the protein sequences. These groups can gain or lose protons depending on the surrounding pH, which alters their charge. Key amino acids that contribute to this are glutamate, aspartate, lysine, arginine, histidine, tyrosine, and cysteine–each with side chains that act as weak acids or bases. The exact pI of a protein therefore depends on its amino acid composition and sequence.

Hydropathy

Hydropathy refers to the distribution of hydrophobic and hydrophilic regions of a protein [26]. It can be measured using the Kyte-Doolittle hydropathy scale which assigns a numerical value to each amino acid based on its hydrophobic or hydrophilic properties and averages these values across the protein sequence.

Hydropathy affects several protein characteristics that can in turn influence antibody developability [26]. The hydrophobic residues tend to cluster in a protein's center and stabilize its core while hydrophilic residues are more common on the outside of the protein in order to interact with aqueous environments. An imbalance of these residues may lead to aggregation and loss of function.

Absolute Charge at pH 7

The absolute charge of a protein at pH 7 refers to the absolute values of the average charge of amino acids at pH 7, derived from the code of the Protein-Sol model [27] and defined as

$$AbsQ = \left| \frac{\sum_{i=1}^{N} Q(r_i)}{N} \right|, \tag{2.1}$$

where $AbsQ$ is the absolute charge at pH 7, N is the number of residues in the sequence and $Q(r_i)$ is the charge of residue r_i in the sequence. In this context, the residues lysine (K) and arginine (R) are positively charged, aspartic acid (D) and glutamic acid (E) are negative, and all other residues have a neutral charge. Protein charge can correlate with hydrophobicity and is one of the primary factors in predicting biophysical properties [28].

Fold Propensity

Fold propensity refers to the tendency of proteins to fold into stable secondary structures. A higher fold propensity increases the probability of forming defined structures, such as α-helices and β-sheets [29]. Proteins with a lack of such structures can often be identified by having a low overall hydrophobicity and a high net charge [30]. Based on these properties, a linear boundary can be fitted to separate disordered proteins from those forming secondary structures. This line has the function

$$R = 2.785H - 1.151, \tag{2.2}$$

where R is the mean net charge and H is the mean hydrophobicity [30]. Rearranging the function gives the fold propensity function as

$$FP = 2.785H - R - 1.151, \tag{2.3}$$

as implemented in the ProteinSol model (see Section 3.3) [27]. The predicted fold propensity correlates inversely with solubility [27].

Disorder

Protein disorder describes regions or entire proteins that lack a stable, regular secondary structure, resulting in increased flexibility within the protein chain [31]. The definition of disorder used in this paper and in ProteinSol (see Section 3.3) is the average propensity of disorder within a protein sequence. This is calculated as the average of the disorder propensities for all residues in the sequence, where each residue is given a propensity value. The disorder function

$$\Omega = \frac{\sum_{i=1}^{N} P(r_i)}{N}, \tag{2.4}$$

where Ω is the average disorder, N is the number of residues and $P(r_i)$ is the propensity value of residue r_i. The propensity values are derived using the following equation:

$$P(r_i) = RC(r_i) - SS(r_i) \tag{2.5}$$

where RC refers to the random coil frequency (i.e, how often an amino acid occurs in a non-secondary structure), and SS refers to the secondary structure frequency (i.e, how often an amino acid occurs in α-helices or β-sheets) [31]. Methods for predicting protein structure can be used as a complementary approach.

Sequence Entropy

Sequence entropy or Shannon entropy is a measure of informational content or variability of amino acids in protein sequences and is defined as

$$SE = -\sum p_i \log_2 p_i \quad \text{for } p_i \neq 0, \tag{2.6}$$

where p_i is the probability of an amino acid occurring in that sequence, calculated as $p_i = c_i / \sum c_j$, where c is the number of occurrences of amino acid i, or j, in the sequence [32]. This measure is important in predicting solubility of a protein, as it has been shown that sequence entropy inversely correlates with solubility [27].

β-strand propensity

β-strand propensity is similar to fold propensity except that it only predicts the tendency of β-strands forming in a protein's secondary structure [33]. These β-strands are a type of secondary stucture element that form the β-sheets.

The algorithm used to predict the β-strand propensity is a simplified Chou-Fasman variation of secondary structure prediction, derived from the code of the ProteinSol [27] model. The Chou-Fasman method works by assigning each amino acid residue a numerical score representing its statistical likelihood to participate in different types of secondary structures such as α-helices or β-strands, based on statistical frequencies observed in a reference set of protein structures [33]. The original propensity values were derived from a small set of proteins in the 1970's and have since been improved using larger sets; the parameters used were taken from [33], using a set of 2168 proteins. The simplified algorithm calculates the average propensity of all amino acid residues in a protein sequence to form β-strands, based on the individual propensities of each residue.

2.2.2 Antibody Binding

Beyond physicochemical characteristics, antibody function is critically shaped by its binding behavior, which determines therapeutic efficacy. Antibody binding describes its ability to interact with a target antigen [6]. This ability can be broken down into binding specificity and affinity which determine how well an antibody binds to its target.

Specificity

Specificity determines an antibody's ability to bind to its intended target while avoiding off-target interactions [6]. Antibodies have specific binding sites (Figure 2.3) that interact with antigens through non-covalent bonds, such as hydrogen bonds and van der Waals forces. These interactions allow antibodies to recognize their targets but factors such as an imbalanced charge distribution can also lead to off-target bindings. This can for example lead to rapid clearance from the body, unwanted immune responses, and unpredictable side effects.

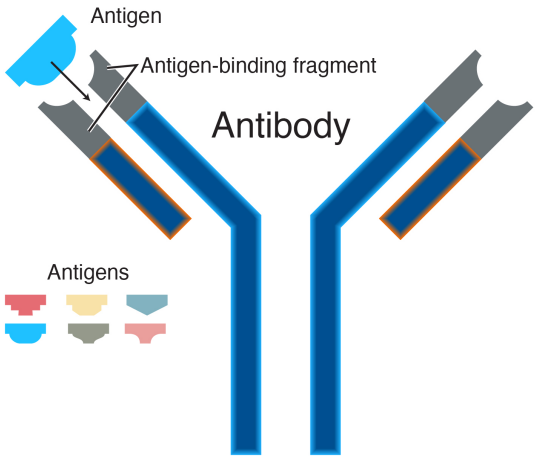


FIGURE 2.3: Shows an antibody's binding site which is specifically designed for one type of antigen [34]. It can be viewed similarly to fitting a puzzle piece (antigen) to the antigen binding site. Antibody specificity allows precise antigen recognition and minimizes off-target effects.

Affinity

The strength of interaction between an antibody and an antigen, called binding affinity, can directly impact the efficacy of an antibody [6]. Higher-affinity antibodies bind more strongly to their target antigen, which ensures sufficient target interactions with low doses. Antibodies can however also be negatively impacted with too high binding affinity. It may lead to off-target interactions, increased risk of aggregation, and prolonged engagement which could have unwanted side effects. Optimizing binding affinity is therefore crucial in ensuring a successful candidate.

2.2.3 Immunogenicity

Immunogenicity refers to the potential of an antibody to cause an immune response when being used as a therapeutic [6, 35]. This can occur when the immune system recognizes the antibody as a foreign substance which can lead to the creation of its own antibodies to neutralize the therapeutic antibody.

An immunogenetic response is more likely to occur if the introduced protein sequences are non-human or misfolded [35]. This reaction can cause a production of anti-drug antibodies (ADAs) which are the immune response's own antibodies used to neutralize the therapeutic antibody. In turn, this will make the drug less effective and accelerate the clearance of it. It can also produce adverse reactions such as inflammatory responses and allergic reactions. These

factors are important to take into account when developing therapeutic antibodies due to its effect on safety and efficacy [6]. Strategies used to minimize antibody immunogenicity include humanization of the antibody by reducing foreign substances as well as identifying potentially problematic areas and modifying them to prevent immune system recognition.

2.3 Machine Learning

Machine learning is a broad term which refers to computational tools used to learn patterns from data and make predictions or for decision making [36]. Several techniques derived from statistics, probability, optimization, and computer science are therefore inherently used to develop these models [37]. Learning patterns from data requires a sufficiently large dataset to capture relevant information and ensure robust model performance. It is also important that the assigned labels of the data are correct in order for the model to predict accurately if labels are provided in the data. Poor-quality data or label noise can lead to misleading conclusions and biased models.

Numerous learning problems and applications leverage machine learning methods [37]. A few examples of these are natural language processing (see Section 2.3.6) and tools in computational biology. These applications usually consist of one or several standard machine learning tasks such as classification, regression, clustering, ranking, and dimensionality reduction:

- **Classification** (see Section 2.3.1): Assigning input data to predefined labels.
- **Regression**: Predicting continuous values from input data.
- **Clustering** (see Section 2.3.5): Dividing data into groups with inherent similarities.
- **Ranking**: Ordering data according to a predefined criterion.
- **Dimensionality Reduction**: Reducing dimensionality and complexity in data while keeping essential information. Often used in computer vision tasks.

Machine learning methods can differ a lot from each other and have varying complexities but they often consist of a similar base pipeline and use similar terminologies [36, 37]. This pipeline will usually consist of data collection, pre-processing, feature extraction, model training, and evaluation. The data, sometimes called examples, are a crucial piece to machine learning models and are the items of interest for a learning algorithm. These inputs are often converted into feature vectors which are numerical representations that capture important attributes of the items. These attributes could for example be a length of a text or a height of a person. Sometimes the examples have corresponding labels which assign the instances into categories [37]. The data used is often split into a training, validation, and test set:

- **Training set**: The dataset used to train the model and adjust its internal parameters. The machine learning algorithm will learn patterns from this dataset and how to associate them with potential labels.
- **Validation set**: Used to tune hyperparameters, the free parameters of the model, and prevent overfitting, i.e, when the model fits too closely to the training data.
- **Test set**: Used to evaluate the model. This dataset is kept separate in order for the model's performance to be assessed on unseen data.

There are many ways to train a machine learning model but the two most common ways are supervised and unsupervised learning [36, 37]. In supervised learning, the training dataset is labeled or assigned to a target value and the goal is to learn how to predict labels or real values from the input features. If the training set is unlabeled, the model has to apply unsupervised learning instead. The goal in this case is to instead find any natural patterns in the data and it is often used in clustering (see Section 2.3.5) and dimensionality reduction. The difficulty with unsupervised learning is that there are no clear metrics to use for performance evaluation and the patterns found might not be easily interpretable.

2.3.1 Classification

Classification refers to the machine learning task of categorizing input by assigning a label to each item [38]. The estimated label y is often calculated using a function $f : \mathbb{R}^n \rightarrow \{1, ..., k\}$, where k is the number of categories. This function will take an input vector $\mathbf{x}$ and assign a label as $y = f(\mathbf{x})$. Classification can also be used to output probability distributions over classes to showcase which classes the input is most likely to belong to. A task consisting of two classes will often use the labels $y \in 0, 1$ and is then referred to as binary classification [36]. If the amount of classes is larger than two, the task is instead a multiclass classification problem.

K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a learning algorithm commonly used for classification tasks [39]. The central idea behind KNN is that similar data points tend to exist in close proximity in the feature space. KNN retains the entire training dataset and makes predictions by comparing new input instances directly to the stored examples. For a given test sample, the algorithm computes the distance, which is typically Euclidean, to all training samples, and identifies the k nearest neighbors, as shown in Figure 2.4. The most frequent class label among these k neighbors is then assigned to the test instance. This majority approach allows KNN to model complex decision boundaries by relying on local data structure.

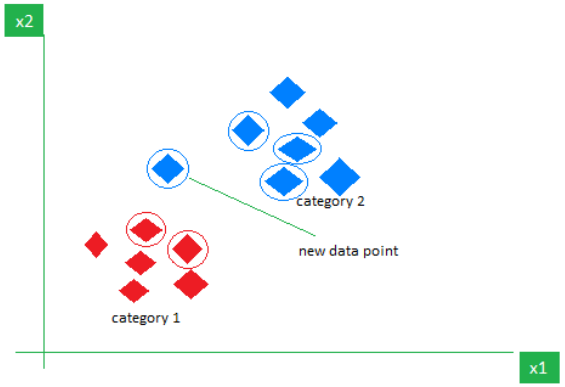


FIGURE 2.4: An example of KNN classification where $K = 5$ [39]. Since the majority of neighbors belong to category 2, the new data point will also be labeled as category 2.

Nearest Centroid

The Nearest Centroid classifier is a supervised learning algorithm that classifies samples based on the proximity to the centroids of known classes in the training data [40]. A centroid is defined as the mean vector of all training samples belonging to a specific class. During training, the algorithm calculates one centroid per class. During prediction, the algorithm assigns each new input to the class whose centroid is closest, using a chosen distance metric, typically Euclidean distance (Figure 2.5). This method is computationally efficient and performs well when classes are approximately linearly separable or form tight, spherical clusters in the feature space. However, it may be less robust in the presence of outliers, since the centroid is influenced by all training samples in the class, potentially skewing the representation of the class center.

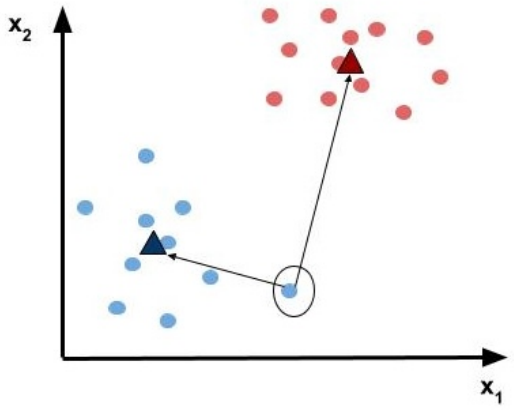


FIGURE 2.5: Illustrates an example of nearest centroid classification. The triangles represent the centroids of two classes: red and blue. The circled instance is a new data point, which is closest to the blue centroid and is therefore classified as belonging to the blue class.

2.3.2 Logistic Regression

Logistic regression is a supervised machine learning algorithm that models the probability of a binary outcome, specifically determining whether or not an event occurs [41]. This makes it suitable for binary classification tasks [42]. An observation consists of features $\boldsymbol{X}$ describing different characteristics of the observation. Assume that observation $\boldsymbol{X}_i$ has a probability P_i of belonging to one of two classes [41]. The aim is to model the probabilities as a linear combination of the features. Therefore, it is necessary to transform the probabilities as they are bounded by 0 and 1, and so the relationship to the features cannot be linearized. The odds of the outcome, defined as $P_i/(1 - P_i)$, express the likelihood of an occurrence relative to the likelihood of a nonoccurrence. Using odds instead of probabilities takes care of the upper bound of the probability, as it goes to infinity when the probability goes to 1. Eliminating the lower bound is done using the natural logarithm of the odds. This is called the logit function and is defined as

$$L_i = \ln\left(\frac{P_i}{1 - P_i}\right). \tag{2.7}$$

The desirable properties of the logit function are that it has no upper or lower bounds and it is symmetric around the midpoint probability of 0.5. Also, the same change in probability gives different changes in logit values depending on the vicinity of the change, when the probability comes closer to 0 or 1, the change in probability translates into a bigger change in the logit value compared to when the probability is closer to 0.5 [41].

With the use of the logit function, it is possible to make a linear combination of the features

relating to the probability. The logit function can then be expressed by the function:

$$L_i = \beta_0 + \beta_1 X_{1,i} + ... + \beta_k X_{k,i} = \boldsymbol{\beta} \cdot \boldsymbol{X}_i, \tag{2.8}$$

where $\boldsymbol{\beta}$ includes the regression coefficients and $\boldsymbol{X}_i$ are the features of observation i, prepended by a one. By taking the inverse of the logit function 2.7, the probability can then be expressed as a function of the features as

$$P_i = \frac{e^{L_i}}{1 + e^{L_i}} = \frac{1}{1 + e^{-L_i}}, \tag{2.9}$$

where the features multiplied by the coefficients are inserted as L_i (Function 2.8).

The maximum likelihood estimation finds the coefficient estimates most likely to give rise to the observed data [41]. The likelihood function (LF) is as follows:

$$LF = \prod P_i^{Y_i} (1 - P_i)^{1 - Y_i}, \tag{2.10}$$

where Y_i (0 or 1) is the observed value for case i and P_i is the predicted probability of that case. The goal is to find values of $\boldsymbol{\beta}$ that give P_i values that maximize the LF. Because probabilities can be small, multiplication of these values can result in very small values that are computationally difficult to handle, LF can be turned into a logged likelihood function (LLF), using the laws of logarithms. The LLF

$$LLF = \ln LF = \sum Y_i \ln P_i + (1 - Y_i) \ln(1 - P_i), \tag{2.11}$$

sums over the logged probabilities instead, resulting in a more stable and efficient process [41].

L_1-regularization

In machine learning algorithms such as logistic regression, L_1-regularization is used to prevent overfitting and improve model generalization by adding a penalty to the model's objective function [43]. This penalty is based on the absolute values of the regression coefficients, encouraging some of them to become zero. The objective function for logistic regression with L_1-regularization is given by

$$Loss = -LLF + \lambda ||\boldsymbol{w}||_1 \tag{2.12}$$

where $||\boldsymbol{w}||_1$ is the L_1-norm of the weights, and λ is the regularization parameter controlling the strength of the penalty. In this context, $\boldsymbol{w}$ represents the vector of coefficients $(\beta_1, \beta_2, ..., \beta_k)$.

The goal is to minimize the loss function in order to find the optimal values of the coefficients $\boldsymbol{\beta}$ that both maximize the log-likelihood and keep the model simpler by penalizing large or unnecessary coefficients. This results in a more robust model by preventing overfitting, especially when the dataset includes irrelevant or redundant features.

2.3.3 Support Vector Machines

Support vector machines (SVMs) are a type of supervised machine learning algorithm that aims to find an optimal boundary to separate data points belonging to different classes [44]. It is often used for classification, the simplest being binary classification, but can also be applied to regression. The model focuses on finding a hyperplane that can separate the data well into their corresponding classes. The hyperplane is a decision boundary which divides the data into respective categories and is often created in a high-dimensional space. Several boundaries may be able to divide the data into the same classes but the optimal hyperplane, which the model aims to find, is the hyperplane that maximizes the margin as in Figure 2.6. The margin refers to the distance between the closest points, also called support vectors, of each of the classes and the hyperplane.

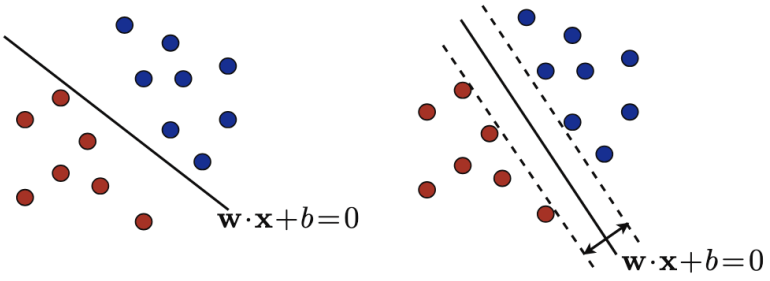


FIGURE 2.6: Two examples of possible hyperplanes in SVM that separate two classes. The figure on the right maximizes the margin [44].

The support vector machines have the task of solving the optimization problem of finding a hyperplane with maximized margin while ensuring that the data points are correctly separated [44]. The hyperplane can be formulated as

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.13}$$

where $\mathbf{x}$ are the data points, $\mathbf{w}$ is the normal vector to the hyperplane, and b is a scalar. Each data point, $\mathbf{x}_i$, will have a label, y_i, associated with it. The labels are $y_i \in \{-1, 1\}$ in the simplest case of linear SVM, representing labels of two classes.

The task of finding a hyperplane with maximized margin can be defined as

$$\begin{gathered}\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \\ \text{subject to the constraint } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i\end{gathered} \tag{2.14}$$

which defines a constrained optimization problem, ensuring that all points are classified correctly with a margin of at least 1 [44]. This task is difficult to solve since the constraints have to be considered during optimization. Using Lagrange multipliers (Section 2.3.3), it is possible to solve the above equation.

Lagrange Multipliers

Lagrange multipliers are a new set of variables used to enforce constraints into an optimization problem [44]. They appear in the Lagrangian function

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] \tag{2.15}$$

where $\alpha_i \geq 0, \forall i$ are the Lagrange multipliers. The first term in the function represents the term to be minimized and is the same as in the above equation 2.14. The second term enforces the constraints with the use of Lagrange multipliers and will act as penalties if the constraints are violated. The task is now transformed into an unconstrained optimization problem which is easier to solve.

The optimal solution is found by calculating the partial derivatives of the Lagrangian with respect to $\mathbf{w}$ and b and setting them to zero [44]:

$$\begin{gathered}\frac{\delta L}{\delta \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \\ \frac{\delta L}{\delta b} = -\sum_{i=1}^{m} \alpha_i y_i = 0 \rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0\end{gathered} \tag{2.16}$$

This results in the Karush-Kuhn-Tucker (KKT) conditions involving stationarity [44]. Another complementary KKT condition is defined as

$$\alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0, \forall i \tag{2.17}$$

which is crucial in order to define if a point is considered a support vector. If $\alpha_i > 0$ then $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ which means that the point is exactly at the margin and is thereby a support vector. If instead $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 1$, then $\alpha_i = 0$ and the point is classified farther from the margin and is not a support vector.

Returning to the optimization problem, $\mathbf{w}$ from equation 2.16 can be substituted back into the Lagrangian (2.15) which results in an optimization problem

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ &\text{subject to: } \alpha_i \geq 0 \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, \forall i \end{aligned} \tag{2.18}$$

that can be solved using standard optimization techniques [44].

2.3.4 Kernel Methods

Kernel methods are widely used techniques in machine learning and enable certain algorithms such as SVMs (Section 2.3.3) to define non-linear decision boundaries [45]. The core idea of the method is to use kernel functions to map data from its original input space to a higher dimensional feature space where linear separation becomes possible. The kernel function computes the inner product between transformed feature vectors, $\Phi(x)$ and $\Phi(x')$, without the need to explicitly compute the transformation itself. This is known as the kernel trick, which significantly reduces computational costs when working with high-dimensional data. Formally, the kernel function K is defined as

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle \tag{2.19}$$

where x and x' are two points, and $\Phi(x)$ and $\Phi(x')$ are the transformed feature vectors.

Radial Basis Function

The radial basis function (RBF) is a commonly used kernel function and is defined for any constant $\sigma > 0$ as

$$K(x, x') = exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \tag{2.20}$$

where σ controls the width of the kernel [45]. The RBF kernel measures the similarity between two points based on their Euclidean distance, where the similarity is the highest when the points

are close and decrease exponentially with increased distance. This kernel is often used in SVMs where it implicitly transforms the data into a high-dimensional space, where a linear decision boundary can be found, even if the data is not linearly separable in the original space.

2.3.5 Clustering

Clustering can be performed as a way to extract meaningful and tangible information from data. There are many different methods of clustering data, this section will look into K-means clustering.

K-means Clustering

K-means is an unsupervised machine learning algorithm used to partition observations into K distinct clusters [46]. The number of clusters, K, is required to be specified by an argument from the user. The algorithm is based on centroids, defined as the mean point of all observations in each cluster, representing the centers of the clusters. The initialization of these centroids can be done using different methods, for example, choosing random points, *k-means++* or letting the user input points. The objective of the algorithm is to minimize J, the sum of the squared error between each data point and the nearest centroids, forming as compact clusters as possible. J of the clusters C can be formulated as follows

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} ||x_i - \mu_k||^2, \tag{2.21}$$

where c_k is cluster k, x_i is data point i and μ_k is the centroid for cluster k.

The K-means algorithm uses the following steps to compute its clusters:

Algorithm 1 K-means Clustering Algorithm

Require: Number of clusters K, data points $x_1, x_2, \ldots, x_n$

Ensure: Cluster assignments and final centroids

Initialize centroids $\mu_1, \mu_2, \ldots, \mu_K$

repeat

for each data point x_i **do**

Calculate distance $d(x_i, \mu_k)$ for all $k = 1, 2, \ldots, K$

Assign x_i to the cluster c_k with the closest centroid μ_k

end for

for each cluster c_k **do**

Update centroid to mean point of its cluster: $\mu_k \leftarrow \frac{1}{|c_k|} \sum_{x_i \in c_k} x_i$

end for

until Centroids converge or maximum iterations reached

Due to the greedy nature of the algorithm, it can converge to a local minimum [46]. To avoid this, it should be run several times, initializing with different centroids each time and choosing the partition with the minimum square error $J(C)$ (Equation 2.21). Choosing the appropriate K is critical for the performance of the algorithm, which can be a problem as the optimal number of clusters is often unknown. Another limitation is that the algorithm uses a Euclidean distance metric, which inherently assumes that clusters are n-spheres. This limits the algorithm's ability to effectively model clusters with non-spherical or complex shapes. Outliers in the data can significantly impact the resulting clusters by "pulling" the centroids, leading to a potential misrepresentation of the true cluster centers. Despite its drawbacks, the K-means algorithm remains the most popular clustering algorithm, credited for its simple implementation and low computational complexity.

2.3.6 Natural Language Processing

Natural Language Processing (NLP) refers to a machine learning field which includes understanding, interpreting, and generating human language [47]. There are several applications in which NLP can be used. These include understanding user queries in a search engine, translating text or speech from one language to another, and chatbots for customer service among others.

Tokenization

Tokenization can be described as the task of dividing a sequence of characters into segments, also called tokens, and is often used as a step in NLP models [48]. A token is a unit that contains

meaningful information and can be used for processing. For an input sentence, tokenization can for example be done by removing characters such as punctuation and white spaces and letting each word be a token. However, the process needs to be tailored according to the relevant application and the same tokenization cannot be used for every implementation. These challenges include being able to segment hyphenated words, processing different language structures, and treating punctuations.

The tokens, which could consist of words, subwords, or characters for example, often need to be converted into numerical embeddings before being processed [49]. These vectors are representations, often of high dimension, and are learned during training. The objective of the numerical embeddings is to capture semantic properties and relationships between tokens. Each token is mapped to a unique fixed-size vector and will contain geometrical properties. Tokens that have similar meanings will reside in close proximity in the high dimensional space which in turn correlates to semantic similarities.

2.3.7 Feature Selection

Feature selection is a method used in many machine learning algorithms, particularly in classification tasks, where the goal is to identify and retain the most informative features while discarding those that are redundant or irrelevant [50]. By reducing the dimensionality of the input space, feature selection can reduce computational cost. In certain cases, it is also appropriate to remove certain features if they contribute to a larger classification error and increases the risk of overfitting [51].

χ^2 Feature Selection

The χ^2 feature selection method is based on the χ^2 distribution, often used in statistics [51]. This statistic quantifies how much the actual distribution of feature values across class labels deviates from what would be expected if the feature and class were independent. A higher χ^2 value indicates a stronger association between the feature and the class, meaning the feature is more informative for classification. Mathematically, this quantity is expressed as

$$\chi^2(t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{2.22}$$

where $e_t \in \{0,1\}$ represents whether the feature is present (1) or absent (0), $e_c \in \{0,1\}$ represents whether the class label is positive (1) or negative (0), $N_{e_t e_c}$ is the observed number of

samples where feature t has value e_t and the class label is e_c, and $E_{e_t e_c}$ is the expected number of such samples under the assumption that the feature and class are independent.

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is a feature selection method that aims to identify the most relevant features by recursively removing the least important ones [52]. It relies on a machine learning estimator which refers to a model or algorithm that can evaluate the importance of features based on the training data. For example, in the case of a linear model like logistic regression, the estimator assigns weights to each feature based on how much it contributes to the model's prediction. Features with smaller weights are considered less relevant.

The RFE process begins by training the estimator on the full set of features [52]. After training, the algorithm evaluates the importance of each feature. The least important feature(s) are removed, and the model is retrained on the remaining subset. This procedure is repeated recursively, eliminating features at each step, until the desired number of features is retained.

Random Forest

Random Forest is an ensemble method that consists of several machine learning algorithms called decision trees [53]. This method can be used for predictions but also provides an inherent feature importance mechanism. The features are ranked using the average decrease in Gini impurity across all trees in the ensemble. The Gini impurity is defined as

$$I_G = 1 - \sum_c \hat{\pi}_c^2 \tag{2.23}$$

where $\hat{\pi}_c$ is the probability that a random entry belongs to class c [54].

Mutual Information

Mutual Information (MI) is a widely used feature selection method [51]. It quantifies the amount of information a given feature provides about the class label meaning that it measures how much knowing the presence or absence of a feature improves prediction. The MI is defined as

$$I(U;C) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} P(U = e_t, C = e_c) log_2 \left(\frac{P(U = e_t, C = e_c)}{P(U = e_t) P(C = e_c)} \right) \tag{2.24}$$

where U is a random variable representing the presence ($e_t = 1$) or absence ($e_t = 0$) of the feature t, C is a random variable representing whether the sample belongs to class c ($e_c = 1$) or not ($e_c = 0$), $\mathrm{P}(U = e_t, C = e_c)$ is the joint probability of observing both feature state e_t and class label e_c, and $\mathrm{P}(U = e_t)$ and $\mathrm{P}(C = e_c)$ are the probabilities of the feature and class occurring independently.

2.3.8 *n*-Fold Cross-Validation

In machine learning tasks where labeled data is limited, splitting the dataset into separate training, validation, and test sets can significantly reduce the amount of data available for training [55]. To address this issue, n-fold cross-validation is widely used as a method for both model selection and performance evaluation.

The core idea behind n-fold cross-validation is to divide the dataset into n equally sized subsets, or folds [55]. The learning algorithm is trained n times, each time using $n - 1$ folds for training and the remaining fold for testing as demonstrated in Figure 2.7. This ensures that every data point is used once as a test example and $n - 1$ times for training, providing a more comprehensive evaluation of model performance.

The choice of n reflects a trade-off between bias and variance [55]. A larger n means each model is trained on nearly the entire dataset, resulting in lower bias but higher variance in the test score, since each test fold is small. A smaller n results in larger test folds and more variance in the training data, which can reduce the variance of the estimated test performance but may increase bias. In practice, values such as 5 or 10 are common for n-fold cross-validation.

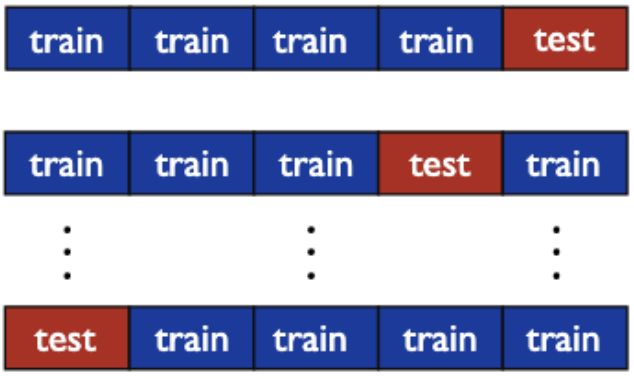


FIGURE 2.7: Shows how the data can be split in 5-fold cross-validation [55]. Each row in the figure shows a different training/test split across the 5 folds.

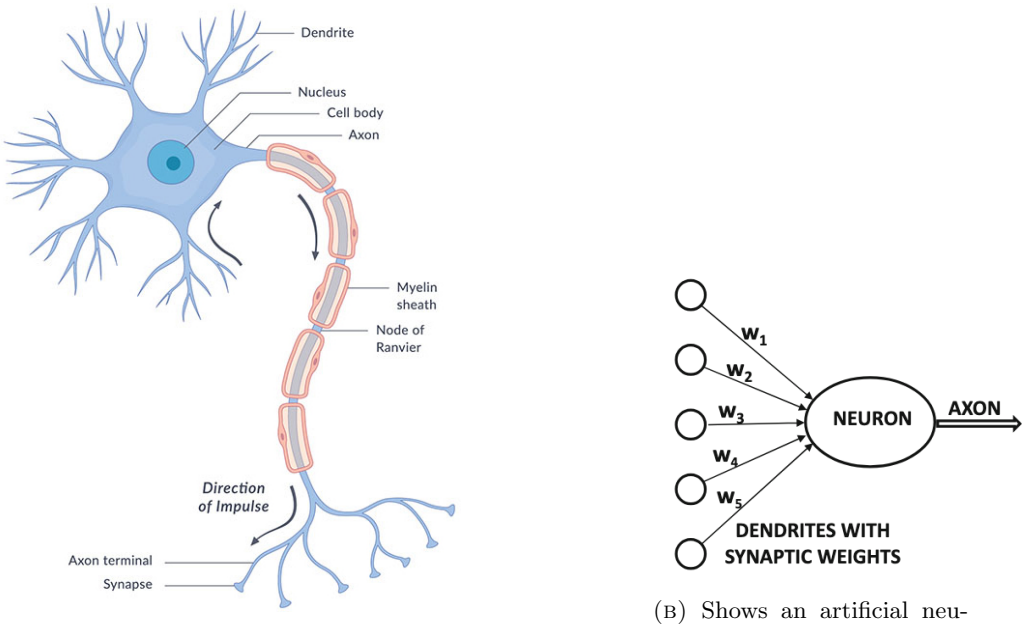
2.3.9 Grid Search

Grid search is a commonly used technique for hyperparameter tuning in machine learning algorithms [56]. It involves systematically searching through a predefined subset, or grid, of hyperparameter values to identify the configuration that yields the best model performance. Each combination in the grid is evaluated, often using cross-validation, to assess how well the model generalizes to unseen data [57].

The main advantage of grid search lies in its conceptual simplicity and exhaustive nature [56]. By evaluating every combination within the grid, it ensures that the best performing configuration within the defined search space is found. This makes it particularly useful as a baseline approach or when the number of hyperparameters is small and the search space is well known. However, as the number of hyperparameters increases, the number of combinations grows exponentially, leading to a significant increase in computational complexity. This issue, commonly referred to as the curse of dimensionality, limits the scalability of grid search.

2.4 Artificial Neural Networks

Deep learning is a field in artificial intelligence that makes use of artificial neural networks (ANNs) [58]. These neural networks are based on the learning mechanisms of human brains and are designed to resemble a similar structure to biological neural networks [59]. Neurons are important cells in the human nervous system and are illustrated in Figure 2.8a. A connection is made between two neurons through one's axons which branch out and connect to another neuron's dendrites. The connection sites are called synapses and is where the information exchange happens. In an artificial neuron, as shown in Figure 2.8b, this process is constructed similarly with neuron-like nodes that can receive weighted input comparable to how biological neurons receive signals in different strengths. The input can then be used in a function and the output sent to other neurons. The frameworks of using ANNs with many neurons to build more complex models with several layers are called deep neural networks and are often considered to be on the forefront of many learning tasks [58].



(A) A simplified representation of a biological neuron.

(B) Shows an artificial neuron.

FIGURE 2.8: Shows a biological neuron to the left [60] and an artificial neuron to the right [59].

The process of learning in an artificial neural network is accomplished by tuning the weights, $w_1, w_2, ..., w_n$, between the neurons [59]. This process is carried out using training data from which the model can predict labels and thereby change weights to improve correctness. Through adjustment of weights and a lot of training data, the ANNs are able to predict more and more accurately and can over time be able to predict unseen data correctly leading to model generalization.

2.4.1 Neural Network Architecture

The base of a neural network is called the perceptron and consists of only one input layer followed by a neuron that provides the output as shown in Figure 2.8b [59]. The perceptron performs binary classification based on the weighted sum of input features and an activation function as described by

$$\hat{y} = \Phi(\bar{\boldsymbol{W}} \cdot \bar{\boldsymbol{X}}^T) \tag{2.25}$$

where $\hat{y}$ is the predicted output, $\Phi(\cdot)$ is the activation function, $\bar{\boldsymbol{W}}$ is the vector of weights, and $\bar{\boldsymbol{X}}$ is the vector of input features. This structure may be expanded into more complex architectures such as multi-layer perceptrons which introduces several layers.

Bias neurons are often included in the layers to help the network model more complex relationships [61]. They add an extra degree of freedom in the learning process. The bias can be an essential part of the algorithm in order to capture an invariant part of the prediction [59]. In certain situations, the output of the model might not be centered around zero, for example in imbalanced binary classification where the feature variables are mean-centered but the mean of the class labels is not zero. In such cases, it's not enough to rely on the weighted sum of input features as the sum of predictions may not match the class distribution. This results in the predicted output being calculated as

$$\hat{y} = \Phi(\bar{\boldsymbol{W}} \cdot \bar{\boldsymbol{X}}^T + b) \tag{2.26}$$

where b is the added bias term that ensures that the prediction can account for the invariant part of the output.

Activation Function

Activation functions are a critical part of the neural network architecture as they define the output of a neuron based on its input [59]. The activation function allows for the model to introduce nonlinearity in order to model complex patterns. The nonlinearity ensures that the model won't collapse into a simple linear regression model.

There are various activation functions that are fit for different problems [59]. A popular activation function is the Rectified Linear Unit (ReLU) function, $\Phi(z) = max\{z, 0\}$, and is used in many modern neural networks. An issue that can arise with the ReLU function are "dying neurons" which refers to when the ReLU function only outputs 0 [61]. This can occur if the weights are initialized as negative while the input values are always non-negative. A solution to the dying neurons is to use a Leaky ReLU activation function instead. This function is defined as

$$\Phi(z) = \begin{cases} \alpha \cdot z & \text{if } z \leq 0 \\ z & \text{otherwise} \end{cases} \tag{2.27}$$

which introduces a new parameter, $\alpha \in (0, 1)$, that ensures that some propagation is possible even for negative z.

The sigmoid activation function is defined as

$$\Phi(z) = \frac{1}{1 + e^{-z}} \tag{2.28}$$

and is another classic activation function that outputs a value in the range (0,1) which can be interpreted as a probability [59]. This makes the function suitable for binary classification problems.

Furthermore, it is common to use the softmax function as an activation function [59]. Given a set of values $v = [v_1, v_2, ..., v_k]$, the softmax function transforms them so that all values are between 0 and 1, and the outputs sum to 1. The function is defined as

$$\Phi(v)_i = \frac{e^{v_i}}{\sum_{j=1}^{k} e^{v_j}} \tag{2.29}$$

where i is the class. This function is useful for multi-class classification, where the model chooses the most likely class based on the highest probability.

Loss Function

The loss function quantifies how far the predicted values are from the actual ground truth values [59]. During training, this function is minimized in order to optimize the model. One of the most simple loss function is the perceptron criterion, defined by

$$L_i = max\{-y_i(\bar{\boldsymbol{W}} \cdot \bar{\boldsymbol{X}}_i^T), 0\} \tag{2.30}$$

where y_i is the observed class label at instance i, $\bar{\boldsymbol{W}}_i$ is the weight vector, and $\bar{\boldsymbol{X}}_i$ is the feature vector [59].

For classification problems, cross-entropy loss is often used as the loss function [62]. The loss functions is defined by

$$L = -\sum_{i=1}^{k} y_i log(\hat{y}_i) \tag{2.31}$$

where y_i is the actual target label and $\hat{y}_i$ is the predicted probability of the instance belonging to the class i.

Multi-Layer Perceptron

The multi-layer perceptron (MLP) is an extension of the perceptron that includes several layers [59]. MLPs contain one or more hidden layers between the input and output layers that perform

computations in order to capture more complex patterns. Each layer contains one or more neurons that are connected to each neuron in the adjacent layer (Figure 2.9), forming a fully connected network.

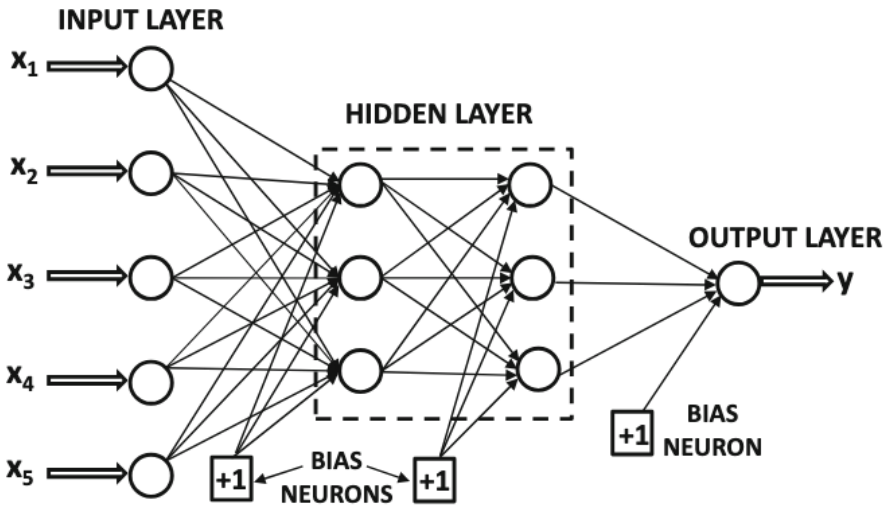


FIGURE 2.9: An example architecture of a Multi-Layer Perceptron (MLP), including bias neurons [59].

In training an MLP, the workflow consists of two primary phases: feedforward and backpropagation, resulting in an epoch [61]. In the feedforward phase, the data is passed through the network, starting at the input layer. At each layer the input is multiplied by corresponding weight matrix and passed through an activation function, $h_{p+1} = \Phi(\boldsymbol{W}_{p+1}h_p + b)$ where h_p is the output from layer p, $\boldsymbol{W}_{p+1}$ is the weight matrix connecting layer p with layer $p+1$, and b is the bias [59]. This continues until the final layer is reached, producing an output probability of the input. The number of hidden layers and the number of neurons per layer are crucial parts of the design of the network and impact the models capacity to learn from the data and capture complex patterns.

Once the data has been passed through the network, the next step is to optimize the network's weights to reduce the error between the predicted output and the actual target [59]. This is where backpropagation is used. Backpropagation is an algorithm used to compute the gradient of the loss function with respect to each weight in the network. The general idea is to minimize the loss by adjusting the weights using gradient descent as defined by

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\delta L_i}{\delta \boldsymbol{W}} \tag{2.32}$$

where α is the learning rate, and $\frac{\delta L_i}{\delta \boldsymbol{W}}$ is the gradient of the loss function L_i at instance i. This update occurs iteratively at every training instance and updates the weights by moving them in the direction of steepest descent of the gradient, effectively minimizing the loss.

In many cases when it comes to MLPs, these gradients can be difficult to compute due to the complexities of the loss functions and is where backpropagation can be used [62]. The propagation algorithm propagates the error from the output layer to the input layer and computes the gradients at each layer step by step. This means that the weight gradients are initialized based on the loss function derivative with respect to the output layer and then computed using the chain rule for each of the hidden layers. This allows the model to update its weights in a way that minimizes the error. Backpropagation removes the need for a closed-form loss function and enables the model to use local information to compute the gradients. While the idea of gradient descent works in theory, there is a challenge in models where the loss has local minima, plateaus, or high-curvature regions (Figure 2.10) [61]. This can make the gradient descent slow or unstable and an optimization algorithm can therefore be needed.

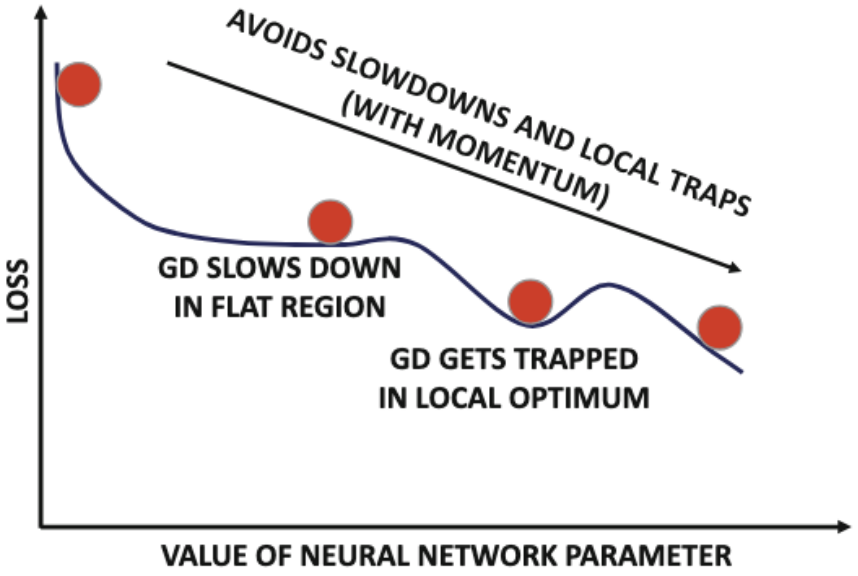


FIGURE 2.10: Shows a simplified visual representation of how the loss surface can vary and how gradient descent (GD) can be affected by plateaus and local optimums [61].

Weight Optimization

Many optimization algorithms used to address the challenge of gradient descent builds on the ideas of momentum [61]. The idea of momentum is that it remembers past gradients and uses this information to help update the weights efficiently. Instead of only pertaining to the current gradient, as in tradition gradient descent, momentum based methods introduce a "velocity"

term that incorporates the previous gradients. This can be conceptualized as a marble rolling down a hill as demonstrated in Figure 2.10. In the steeper parts, the marble gains speed which enables it to escape local minima or flat regions even though the current gradient is small or the opposite sign. This approach helps to smooth out the updates and reduces the risk of oscillation in certain parts of the loss surface.

A popular optimization algorithm that incorporates momentum is Adaptive Moment Estimation (ADAM) [61]. ADAM introduces momentum using an exponentially smoothed moving average of the gradients

$$F_i = \rho_f F_{i-1} + (1 - \rho_f)\nabla \boldsymbol{W}_i \tag{2.33}$$

where F_i is the smoothed gradient and ρ_f is the decay parameter which is used to control the rate of which the previous gradients affect the current update. Using the smoothed gradient, F_i, the weight update is carried out as in

$$\boldsymbol{W}_i = \boldsymbol{W}_{i-1} - \frac{\alpha_i}{\sqrt{A_i} - \epsilon} F_i \tag{2.34}$$

where α_i is the initial learning rate, A_i is an exponentially smoothed version of squared gradients, and ϵ is a small constant added to avoid dividing by zero [61]. The term A_i is in this case used as a normalizing factor to help avoid excessively large steps and is computed as

$$A_i = \rho A_{i-1} + (1 - \rho)(\nabla \boldsymbol{W}_i)^2 \tag{2.35}$$

where ρ is a decay factor typically close to 1.

2.4.2 Neural Network Optimization

There are several techniques that can be applied to improve the performance of neural networks. The following sections showcase some of the methods that can be applied to MLPs to optimize their architecture.

Early Stopping

Early stopping is a technique used in the training phase of a neural network in order to prevent overfitting [63]. It works by monitoring the performance of the model on a validation set through

its loss (Section 2.4.1). During training, as the model minimizes loss on the training data using gradient descent, the loss on the validation set typically reduces too, up to a point. Eventually, the validation loss may stop improving and start increasing, even though the training loss continues to decrease. This indicates that the model is starting to memorize the training data, meaning that it's starting to overfit, instead of learning patterns that generalize. Early stopping halts training at the point just before validation loss begins to rise, which helps retain a model that generalizes well.

Learning Rate Decay

Reducing the learning rate during neural network training, commonly known as learning rate decay, is a widely adopted strategy that enhances both optimization and generalization [64]. Starting with a large learning rate allows the optimizer to take substantial steps in the loss surface. This facilitates escaping plateaus or local minima (Figure 2.10), enabling the model to explore more promising regions of the parameter space. An initially high learning rate prevents the network from overfitting to noisy or less informative patterns in the early stages of training. As the learning rate decays, the model becomes more sensitive to complex and meaningful patterns, improving its ability to generalize. Decaying the learning rate over time helps the optimizer settle into a minimum by taking smaller, more precise steps. This reduces oscillations around the minimum and leads to a more stable convergence. Learning rate decay balances the need for rapid initial learning with the precision required for fine-tuning, leading to models that are both accurate and generalizable.

L_2-regularization

L_2-regularization is a technique used to reduce overfitting in neural networks by penalizing large weights in the hidden layers [63]. Instead of directly limiting the number of weights in a model, L_2-regularization softly discourages large weight values by adding a penalty term to the loss function defined as

$$J = L + \lambda \sum_i w_i^2 \tag{2.36}$$

where L is the original loss as described in Section 2.4.1, λ is the regularization strength, and w_i are the individual weights in the model.

L_2-regularization biases the model toward simpler, smoother functions, which are less likely to overfit the training data and more likely to generalize to new examples [63]. It also prevents individual neurons in the hidden layers from having excessive influence, which stabilizes training and leads to more robust representations.

Batch Normalization

Batch normalization (BN) is a technique that helps to stabilize and accelerate the training of a neural network [61]. As neural networks grow deeper, gradients during backpropagation can become very small or very large, called vanishing and exploding gradients respectively. This can make learning unstable or slow. Batch normalization mitigates this by normalizing the inputs to each layer, ensuring that activations remain within a controlled range. This makes gradient flow more consistent, leading to faster and more stable convergence during training.

During training, parameters of earlier layers change constantly, which causes the input distribution to subsequent layers to keep shifting. This internal shift slows down learning because each layer has to adapt to new input distributions repeatedly. BN reduces this shift by normalizing layer inputs, helping each layer learn more independently and reliably.

Dropout

Dropout is a regularization technique that helps prevent overfitting in neural networks by injecting randomness into the training process [63]. During training, dropout randomly removes nodes from the input and hidden layers of the network. When a node is dropped, all its incoming and outgoing connections are also removed. This results in a different subnetwork being sampled for each training step, effectively simulating the training of many different neural networks. While dropout samples many subnetworks, it doesn't train them separately. Instead, all subnetworks share weights. Every smaller batch update applies to a different sampled network, but the weight updates are accumulated in the shared parameters.

The randomness introduced by dropping nodes forces the network to learn more robust features [63]. It prevents co-adaptation, where neurons rely too heavily on specific peers. Since neurons cannot rely on any one other neuron always being present, they learn to generalize better.

2.4.3 Transfer Learning

Transfer learning is a technique used in machine learning applications where a model trained on one task is adapted to perform a different, but related task [65]. The method is particularly useful when the new task has limited amount of training data, as it allows the model to leverage previously gained knowledge. It is also beneficial for when labeled data is scarce, difficult, or expensive to acquire since transfer learning allows the reuse of feature representations, model parameters, and in some cases pre-trained models. This removes the need to build a model from

scratch which can be costly and time-consuming.

Deep learning models benefit greatly from transfer learning as large amounts of data are often required for deep neural networks to generalize well [66]. This process often entails feature extraction and fine-tuning. Feature extraction involves retrieving useful feature embeddings from new data using the pre-trained model while fine-tuning consists of updating some layers of the model while training on the new task. Transfer learning is often applied to natural language processing (Section 2.3.6) where pre-trained models have already processed massive text datasets and can be fine-tuned for tasks such as question answering.

2.5 Evaluation of Machine Learning Models

There are several metrics that can be computed in order to evaluate a classifier [67]. The following sections will be based on the binary classification problem though it is possible to expand these computations to multi-class classification.

Confusion Matrix

Acquiring a simple overview of a binary classifier's performance can be done through a confusion matrix as shown in Figure 2.11. The matrix consists of values for true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN). These correspond to which values were correctly or incorrectly classified as positive and negative respectively [67]. For example, if a protein is soluble and the model predicts it as soluble, this counts as a true positive.

		Predicted	
		Positive	**Negative**
Actual	**Positive**	# True Positive (TP)	# False Negative (FN)
	Negative	# False Positive (FP)	# True Negative (TN)

FIGURE 2.11: Shows a confusion matrix.

Accuracy

Accuracy is defined as the ratio of correct predictions considering all predictions [67]. Letting $n = TP + TN + FP + FN$, accuracy can be expressed as

$$Accuracy = \frac{TP + TN}{n} \tag{2.37}$$

However, accuracy can be misleading, especially in imbalanced datasets [68]. If, for example, a dataset has a high majority of negatives and a model predicts only negatives, the model will still have a high accuracy. It is therefore important to know the importance of misclassification between classes and use other evaluation metrics.

Precision

Precision is a measure of the ratio between true positives considering all predicted positives [67]. A precision near 1 indicates that most predicted positives are correct while a value near 0 instead indicates that the model results in a lot of false positives. The precision is calculated as

$$Precision = \frac{TP}{TP + FP} \tag{2.38}$$

Precision can be particularly important when using imbalanced datasets [68]. An example could be detecting a rare disease where there are a lot of negatives. In this case it is important that if a disease is detected, it is highly accurate.

Recall

Recall is defined as

$$Recall = \frac{TP}{TP + FN} \tag{2.39}$$

and measures the ratio of predicted positives out of all the actual positives [67]. A high recall value (close to 1), indicates that the model is able to correctly predict most of the positive while a lower recall value, close to 0, shows that there is an issue with a large ratio of false negatives.

Similarly to precision, the metric is important when the data is imbalanced [68]. Using the same example as before with detecting a rare disease, it is important to ensure that as many instances of the disease as possible are actually detected.

F_1-score

The F_1-score is advantageous to use for imbalanced data and represents the harmonic mean of precision and recall [67]. It is defined as

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.40}$$

The F_1-score is often used for when the false negatives and false positives are equally important and the problem is therefore symmetric. The score balances the trade-off between precision and recall and provides a reliable metric when the goal is to achieve a balance between correctly identifying positives and minimizing false positives [68].

Area Under Receiver Operating Characteristic Curve

In a Receiver Operating Characteristic (ROC) curve (Figure 2.12), the true positive rate (recall) is plotted against the false positive rate for different threshold values, i.e the threshold of the prediction for which the point is considered a positive [67]. This implies that with a threshold of 1, the model classifies all samples as negative ((0,0) on the ROC curve), and for a threshold of value 0, all points will be predicted as positive ((1,1) on the ROC curve). A perfect classifier would always predict the positives correctly and therefore reach the upper left corner of the graph. Meanwhile, if the classifier predicts randomly, the ROC curve would display a straight diagonal line which indicates that there is an equal probability for the point to be predicted as positive correctly as incorrectly.

To measure a classifier's performance using the ROC curve, the area under the curve (AUC) can be calculated [67]. As is illustrated in Figure 2.12, a perfect classifier would result in an AUC value of 1 while a classifier that predicts randomly would result in an AUC value of 0.5.

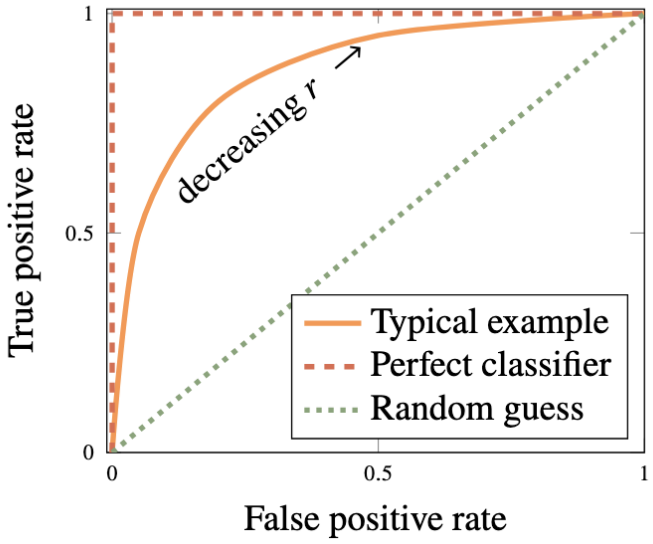


FIGURE 2.12: Shows a Receiver Operating Characteristic (ROC) curve [67].

Matthews Correlation Coefficient

Matthews Correlation Coefficient (MCC) is a score that takes into account all four rates from the confusion matrix and can thereby be considered to be a better measure of performance than F_1-score for example [69]. It is defined as

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{2.41}$$

and its value ranges from -1 (total disagreement) to +1 (perfect prediction), with 0 indicating no better than random guessing.

CHAPTER 3

Existing Prediction Models

3.1 NetSolP

NetSolP is a language model (see Section 2.3.6) based architecture that is built to predict solubility of amino acid sequences [70]. By treating the protein sequences similar to sentences within a language model framework, NetSolP captures sequence patterns indicative of protein solubility. The architecture leverages ensembles of variations of two publicly available Evolutionary Scale Modeling (ESM) (see Section 3.1.2) models : ESM12 and ESM1b, which are based on the Transformer model (see Section 3.1.1). These models, containing 12 layers (84M parameters) and 33 layers (650M parameters) respectively, provide pre-trained embeddings that capture contextual relationships between amino acids. The integration of these models enables NetSolP to generate robust solubility predictions by utilizing the deep contextual knowledge of protein sequences.

3.1.1 Transformer Model

The Transformer model is a deep neural network architecture designed for sequence modeling [71]. Unlike previous sequence models such as Recurrent Neural Networks (RNNs), the Transformer relies entirely on self-attention mechanisms and feed-forward layers, eliminating the need for recurrence or convolution operations. This enables the model to capture both short- and long-range dependencies between tokens.

Transformers consist of an encoder-decoder structure [71]. The encoder processes an input sequence, such as a sentence, and generates contextual embeddings that represent the meaning of the input. The decoder subsequently uses these embeddings to produce an output sequence, such as a translated sentence. The Transformer model does not inherently account for sequence order and so the positional information is preserved by adding positional encodings to the input embeddings. This allows the model to retain sequence order effectively.

Encoder

The encoder is composed of multiple layers, each containing two sub-layers: a multi-head self-attention (see Section 3.1.1) layer and a feed-forward layer (Figure 3.1) [71]. The feed-forward layer consists of two linear transformations with a ReLU activation function applied between as shown in

$$FFN(\boldsymbol{x}) = ReLU(\boldsymbol{x}\boldsymbol{W_1} + b_1)\boldsymbol{W_2} + b_2 \tag{3.1}$$

where $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are weight matrices, and b_1 and b_2 are bias terms. Residual connections, where the input is added back onto the output, and layer normalization are applied around each sub-layer to enhance training stability and gradient flow.

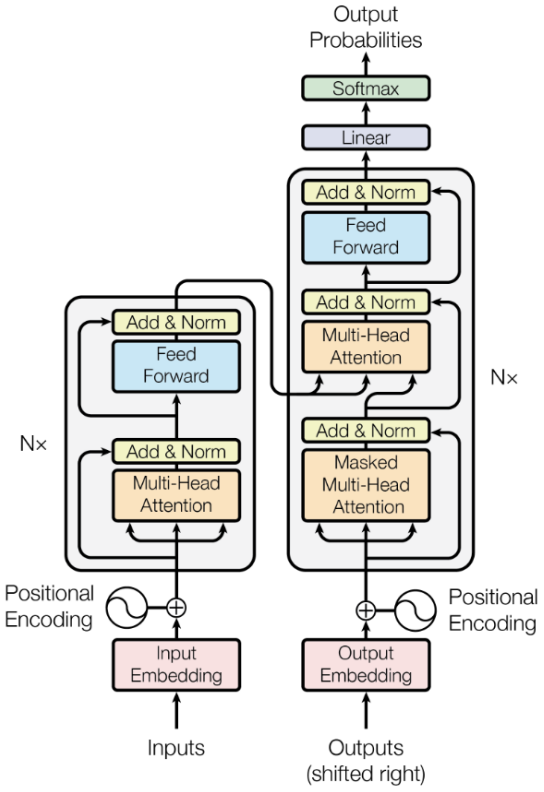


FIGURE 3.1: Shows the encoder-decoder architecture for the Transformer model [71]. The left part represents the encoder and the right the decoder.

Multi-Head Self-Attention

The self-attention mechanism allows the model to capture dependencies between tokens within an input sequence [72]. For each token embedding, three vectors are computed: Query ($\boldsymbol{Q}$), Key ($\boldsymbol{K}$), and Value ($\boldsymbol{V}$). The vectors are obtained by multiplying the embedding with learned weight matrices. These matrices are optimized during training.

The attention score determines the importance of a token relative to others [71]. It is computed by comparing the token's query with the keys of the other tokens in the sequence using the scaled dot-product attention equation

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{3.2}$$

where d_k is the dimensionality of $\boldsymbol{Q}$ and $\boldsymbol{K}$, used for scaling. This score indicates how much each token should pay attention to other tokens.

Multi-head self-attention extends this mechanism by running multiple attention heads in parallel, enabling the model to capture diverse relationships in the sequence [71]. Several attention-heads can provide outputs using different weights and learning different patterns from the sequence. This means that several queries, keys, and values are computed and the attention scores are calculated in parallel. Once the scores are computed, they're concatenated into a single vector

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Concat(head_1, ..., head_h)\boldsymbol{W}_O \tag{3.3}$$

and passed through a linear layer which combines the learned information into a final representation. In the equation, $\boldsymbol{W}_O$ represents the output weight matrix and h is the number of attention heads.

Decoder

The decoder is responsible for generating an output sequence from the embeddings [71]. It is built similarly to the encoder but has an added layer in the start of each block (Figure 3.1). The added layer is a masked multi-head attention layer, which restricts attention to past tokens by masking future positions, ensuring that the model does not attend to information it has not yet generated. This ensures that the model does not cheat by looking ahead. At the end of the decoder pipeline, a linear transformation followed by a softmax function is added to convert the output into predicted probabilities for next-token generation.

3.1.2 Evolutionary Scale Modeling

Evolutionary Scale Modeling (ESM) refers to a series of transformer-based (see Section 3.1.1) models that are designed to capture evolutionary, structural, and functional properties of proteins through unsupervised learning [73]. The central idea of ESMs is to leverage large datasets of protein sequences and apply transformer architecture to learn rich, high-dimensional representations that reflect the biological characteristics of the protein. The ESMs employ the encoder structure of the transformer model with the aim of creating information-rich embeddings.

ESMs are trained with the objective of masked language modeling [73]. During training, a portion of the amino acid sequence is randomly masked or "hidden", and the model is tasked with predicting the masked residues based on the surrounding context. In this way, the model is forced to learn meaningful representations and dependencies across amino acids at varying distances. The model learns not only the properties of the individual amino acids but also how they interact and contribute to the overall structure and function.

The training data for the ESM models consist of 86 billion amino acids from a diverse set of 250 million sequences [73]. The dataset was curated to include a wide range of protein families and species, ensuring that the model learns generalizable features applicable to various biological contexts. The large dataset is pivotal for training models that need to capture a wide variety of protein characteristics.

ESMs are not just limited to predicting sequence-related properties [73]. They can be fine-tuned for a variety of tasks, such as predicting protein function, and other biological properties. Since these models are trained in an unsupervised manner, they are capable of transferring their learned representations (see Section 2.4.3) to a wide range of biological tasks, even when the available labeled data for a specific task is limited.

3.1.3 Training

The NetSolP architecture was trained on a dataset consisting of 69,420 proteins from the TargetTrack database, a part of the Protein Structure Initiative (PSI) [70]. The model is built upon fine-tuned pre-trained ESMs by adding a linear classification layer onto the pre-existing embeddings. The output from the pre-trained model is a single vector representing the entire protein, created by averaging the embeddings of individual amino acids. The vector can then be used as input to the linear classification layer which is a feed-forward layer that maps the high-dimensional output of the embeddings to a probability value. NetSolP employs end-to-end fine-tuning meaning that the entire transformer model, including all layers and parameters, were updated during training on the solubility dataset which allows better adaptation to solubility predictions. The learning rates were set to 3×10^{-6} for the pre-trained model and 2×10^{-5} for the classification layer to avoid disrupting the pre-trained embeddings while still optimizing efficient learning. Early stopping was used to terminate training when performance on a validation set plateaued, preventing overfitting of the model.

3.1.4 Prediction

To predict solubility using the NetSolP model, the input data must be in a *FASTA* format containing sequence identifiers (SIDs) followed by the protein sequences as shown in Figure 3.2 [74]. To ensure computational efficiency, the input data is truncated to a maximum length of 510 amino acids during training and 1022 amino acids during prediction.

```
> Abagovomab
QVKLQESGAELARPGASVKLSCKA...
```

FIGURE 3.2: Shows an example snippet of input data in a FASTA format. *Abagovomab* is an example of a protein sequence identifier (SID) and the following sequence of letters refers to the protein sequence where each letter represents an amino acid.

The input sequences are firstly converted into tokens using a predefined alphabet, *ESM12_alphabet.pkl*, which transforms each amino acid in the sequence to a numerical value [74]. This allows the model to process protein sequences in a numerical format. The corresponding numerical values for each token can be seen in Figure A.1. The tokenization is followed by mapping the sequences into high-dimensional embeddings, capturing 768 features for each token. A positional encoding is also added at this stage to include positional information of the tokens.

The embeddings are then passed on to multiple stacked encoder layers (Figure 3.3), similar to those of the Transformer model encoder (see Section 3.1.1) [74]. The blocks consist of multi-head self-attention followed by a feed-forward layer. The depth of the models are 12 and 33 stacked layers for *ESM12* and *ESM1b* respectively, providing two models of differing complexities.

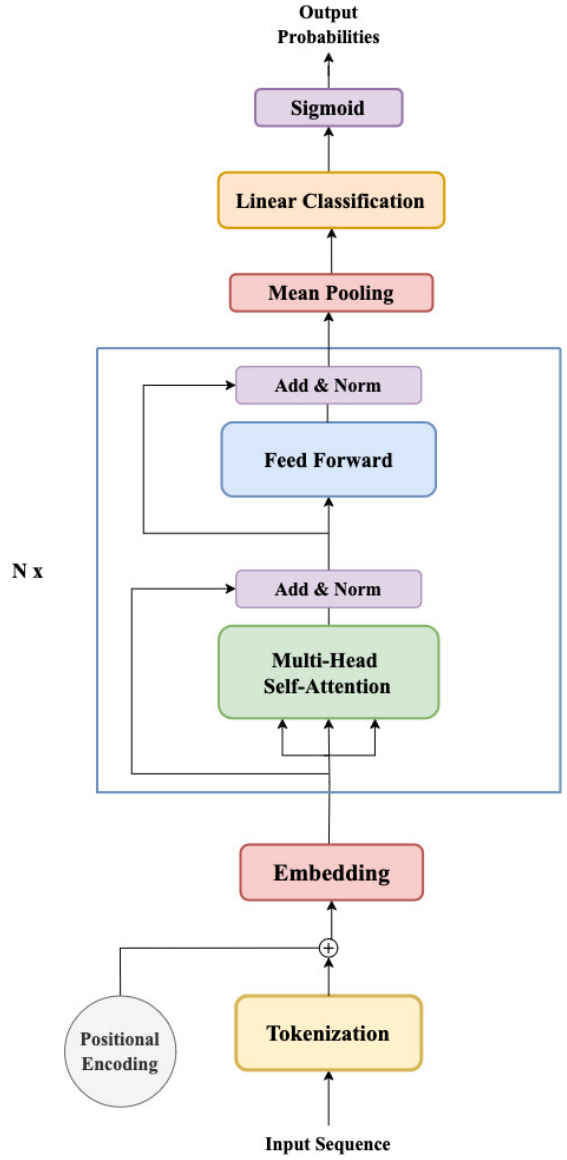


FIGURE 3.3: Shows a flowchart of the NetSolP model architecture. The multi-head self-attention layer, represented by the blue outline, can be repeated N number of times. In the case of *ESM12* and *ESM1b*, the multi-head self-attention layer was repeated 12 and 33 times respectively.

Once the embeddings have been processed by the stacked self-attention layers, a mean pooling is applied [70]. The amino acid feature vectors are averaged across the sequence creating a single vector with 768 features that represent the entire sequence [74]. The pooled vector can then be passed through a linear classification layer consisting of a general matrix multiplication (GEMM). This layer outputs a single logit value per protein sequence, which is subsequently passed through a sigmoid activation to yield a probability of solubility. Finally, this logit value is passed through a sigmoid activation function, converting it to a probability score between 0 and 1, indicating the likelihood of protein solubility.

The output from the NetSolP model is provided in a *CSV* file containing SID, *FASTA* (protein sequence), and results from the ensemble models (Figure 3.4) [74]. The amount of results depend on how many models are used from the ensemble and the final solubility prediction is computed as the average of probabilities from all ensemble models.

```
sid, fasta, solubility_model_0, solubility_model_1, ..., predicted_solubility
Abagovomab, QVKLQESGAELARPGASVKLSCKA..., 0.2223, 0.310, ... , 0.2615
```

FIGURE 3.4: Shows an example snippet of output from the NetSolP model in a CSV format. The output contains the sequence identifier (SID), the protein sequence where each letter refers to an amino acid, and the probability of the protein being soluble for each applied model as well as an average of all models.

3.2 Solubility-Weighted Index

The Solubility-Weighted Index (SWI) is a computational method (Figure 3.5) used to predict protein solubility by analyzing the relationship between amino acid composition and structural flexibility [21]. The method is based on the observation that structural flexibility (see Section 2.2.1) correlates with protein solubility. The SWI is calculated as an arithmetic mean of assigned solubility weights of the amino acids in the sequence and is defined as in the equation

$$SWI = \frac{1}{N}\sum_{i=1}^{N} W_i \tag{3.4}$$

where N is the number of residues in the sequence and W_i is the optimized weight of residue i. The solubility probability can then be calculated using a logistic regression model as shown in the following equation

$$\text{probability of solubility} = \frac{1}{1 + exp(-(a \cdot SWI + b))} \tag{3.5}$$

where $a = 81.05812$ and $b = -62.7775$. The constants a and b were derived from fitting the logistic regression to the PSI: Biology (see Section 4.1) dataset which consisted of 11,226 protein sequences with solubility labels. The goal of the fitting was to find the values of a and b that minimize the difference between the predicted solubility probability and the actual solubility labels for the training data using the calculated SWI for the protein sequences in the PSI: Biology dataset.

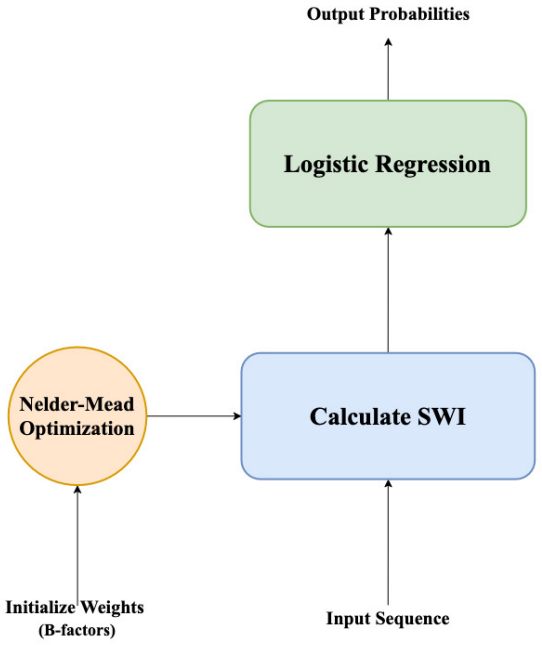


FIGURE 3.5: Flowchart illustrating the steps in the SWI model, including how B-factors are used to compute solubility weights and predict solubility probability. B-factors (see Section 2.2.1) refers to the values used to measure protein structure flexibility.

3.2.1 Amino Acid Solubility Weights

To compute the solubility weights for each amino acid, a dataset of 12,216 protein sequences from the TargetTrack database was used [21]. The proteins were then clustered at 40% identity into 5050 non-redundant clusters using USEARCH to prevent overfitting due to sequence homology. This means that protein sequences that shared at least 40% of their amino acids in the same order were grouped together in a cluster. A dataset consisting of 3,173 protein sequences with solubility data from the eSOL database as well as "stickiness" dataset consisting of 397 proteins with surface accessibility information were used for validation.

The amino acid solubility weights were then initialized from normalized B-factors (see Section 2.2.1) and optimized through a 10-fold cross-validation approach [21]. The goal was to maximize the AUC-ROC metric by optimizing the amino acid weights using the Nelder-Mead optimization algorithm (see Section 3.2.1). The optimization was carried out 1000 times for each sample, each time using bootstrap resampling of 1000 soluble and 1000 insoluble proteins, a method that samples with replacement. This resulted in 1000 sets of optimized weights which were then used to calculate a mean weight for each amino acid.

Nelder-Mead optimization

The Nelder-Mead optimization method is a derivative-free method used to obtain an optimal value of a function [75]. The method is initialized with a simplex which is a set of $n+1$ points in a n-dimensional space. These points represent candidate solutions in the parameter space to be optimized. With each iteration, the simplex is transformed by modifying the worst performing point. The transformation is done through four steps, reflection, expansion, contraction, and shrinkage. Reflection is done by updating the worst point with a potentially better guess which is the point reflected in the centroid of the simplex. If this improves the overall objective, the point is expanded by moving it further away from the centroid in the same direction as before. If reflection fails, contraction is carried out by instead moving the point closer to the centroid, reducing the search space and moving it closer to more promising points. If the contraction also fails, shrinkage is done by reducing the size of the simplex by updating all points closer to the best point. The algorithm is stopped when the simplex converges.

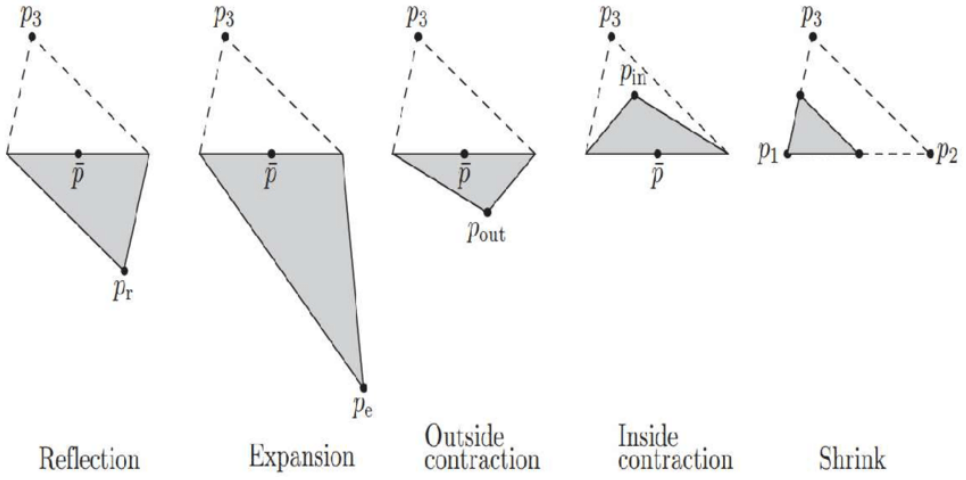


FIGURE 3.6: Shows the steps of Nelder-Mead optimization where p_3 is the worst performing point [76]. From left to right: reflection, expansion, outside contraction, inside contraction, and shrinkage.

3.3 ProteinSol

ProteinSol is a model that uses multiple descriptors relating protein sequences to features that affect solubility [27]. The development of the ProteinSol model began by computing 35 protein features from sequences in a dataset eSOL (3,173 samples) [77], creating an initial fit to the solubility data. This initial fit serves as a baseline to understand how each feature correlates to the solubility of proteins [27]. The features are the 20 amino acid compositions, 7 amino acid composites: K-R, D-E, K+R, D+E, K+R-D-E, K+R+D+E, F+W+Y and 8 predicted features: sequence length, pI, hydropathy, absolute charge at pH 7, fold propensity, disorder,

sequence entropy, and β-strand propensity (see Section 2.2.1). The amino acid composites are combinations (represented by +) or differences (represented by -) of specific amino acid counts that are believed to correlate to protein solubility. The dataset utilized contains solubility labels for each protein, measured in experiments [77]. To identify which features most effectively predict solubility, the model uses two extreme subsets of the dataset: proteins whose solubility falls within the highest and lowest 5% of solubility values, meaning that those subsets are both of size 159 [27].

3.3.1 Z-scores

A z-score is the number of standard deviations a data point is from the mean value, calculated as

$$z = \frac{x - \mu}{\sigma}, \tag{3.6}$$

where x is the data point, μ is the mean of population and σ is the standard deviation of the population. The model calculates z-scores for the proteins in the extreme subsets and then uses a $ZScoreDiff$ value for each feature that describes its efficiency in differentiating between low and high solubility; this is derived from the code of the model [27]. The $ZScoreDiff$ for feature i is calculated as

$$ZScoreDiff_i = \boldsymbol{z}_{top_i} - \boldsymbol{z}_{low_i}, \tag{3.7}$$

where $\boldsymbol{z}_{top_i}$ is the mean of the z-scores for feature i in the higher subset and $\boldsymbol{z}_{low_i}$ is the mean of the z-scores for feature i in the lower subset. The weights are then the absolute value of the $ZScoreDiff$, normalized to sum to 1 as in

$$W_i = \frac{|ZScoreDiff_i|}{\sum_{i=1}^{N} ZScoreDiff_i}. \tag{3.8}$$

3.3.2 Feature Selection and Significance

By analyzing weight magnitudes and using correlation techniques, the contribution of each feature to the prediction of solubility was assessed, and the features with low contributions were eliminated [27]. This was done until the performance of the model dropped. The final model includes 10 features: H, L, V, K-R, D+E, F+W+Y, length, absolute charge, fold propensity and sequence entropy, with a correlation coefficient of 0.621 between experimented and calculated values.

3.3.3 Solubility Prediction Algorithm

In the final solubility predicting algorithm, the model computes all features, but it only utilizes those deemed to significantly contribute to the prediction accuracy, derived from the code of the model [27]. The feature values are checked to ensure that they fall within the high and low baseline subsets' averages, otherwise the values are changed to the closest subset average. The values are then linearly interpolated between the high- and low-solubility means, meaning that the feature value is mapped proportionally to a scale where the low-solubility average corresponds to 0 and the high-solubility average corresponds to 1. They are then multiplied by their corresponding weights, summed, and scaled between 0 and 1. A flowchart of the computational process can be seen in Figure 3.7.

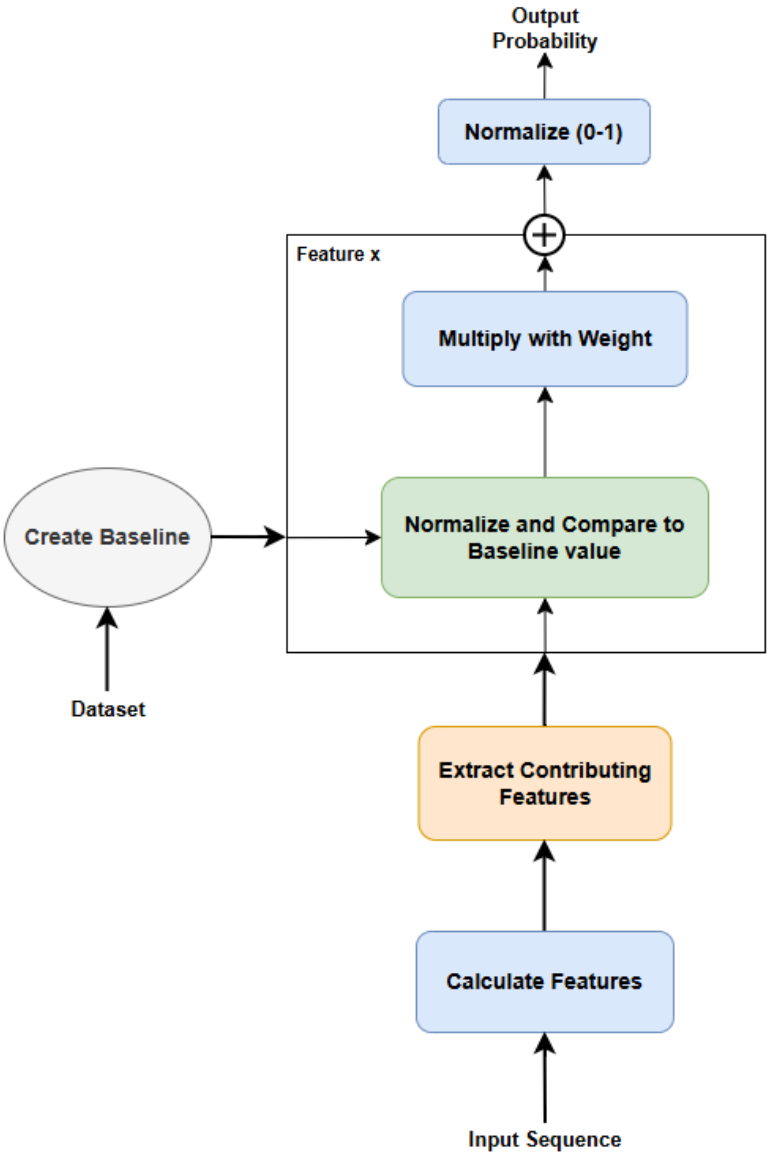


FIGURE 3.7: Flowchart illustrating the steps in the ProteinSol model. The square block is repeated for each feature.

CHAPTER 4

Method

4.1 Data

Several datasets containing protein sequences were used in this project. The existing models required only an SID identifier and a corresponding protein sequence to make predictions—this was the minimum requirement for dataset inclusion. To evaluate model performance, some datasets also needed ground truth labels, such as soluble or insoluble. While many protein and antibody datasets meet the basic criteria, few include experimentally verified solubility labels due to the difficulty of producing them. Therefore, datasets were primarily selected from those used in existing solubility prediction models.

The PSI: Biology dataset [78] was retrieved from the NetSolP open-source code [74]. It consists of 11,226 proteins, expressed in E. coli, with solubility labels, and was part of the training set for NetSolP. The protein data were originally curated from the TargetTrack [79] database, created by the Protein Structure Initiative (PSI) between 2000–2017 to expand protein structure data. Solubility labels were later added by other participants.

The NESG dataset [78], with 1,323 protein sequences expressed in E. coli, was also retrieved from NetSolP's code. Used as a validation set in that model, it included solubility labels similar to PSI: Biology. While the sequences also came from TargetTrack, the solubility annotations came from different contributors.

A third dataset, eSOL, was obtained through the open-source code of the SWI model [80]. Originally curated using in vitro expression from E. coli gene segments, solubility was assessed through centrifugation [77]. The original study showed that the distribution of solubility probabilities was bimodal, with peaks corresponding to proteins classified as aggregation-prone (<30% solubility) and highly soluble (>70% solubility). A threshold of 0.5 was therefore used in this project to binarize the data for classification, allowing the use of the full dataset while maintaining a reasonable midpoint between the defined extremes. After further pre-processing, by removal of duplicates and entries with missing ground truth, the dataset finally included 3,173 protein sequences with solubility as labels.

The datasets NESG, eSOL, and PSI: Biology were also combined into a pooled dataset referred to as *Superset* in this report. This provided a larger dataset consisting of 15,722 samples with solubility labels. The increased data volume is beneficial for machine learning models, as it can improve generalization and enable the use of more complex algorithms that perform better with larger training sets.

Further, a dataset called Thera-SAbDab [81] was used as proposed by the partnering company, Bionamic AB. This dataset included 1,074 heavy and light chain sequences from antibody proteins. The full SAbDab database contains all antibody data from the Protein Data Bank (PDB) [82] with corresponding annotations and Thera-SAbDab is a subset containing therapeutic antibodies. This dataset did not include any ground truth solubility labels.

A dataset called PDBSol was retrieved from a prediction model, ProtSolM [83]. The model itself did not end up being used in this project but it provided a large dataset consisting of 64,598 proteins expressed in E. Coli with corresponding solubility labels. The dataset compiled protein sequences from several protein databases including:

- **UniProtKB** - a large collection of high-quality, manually selected protein sequences. PDBSol included 4,337 soluble proteins from this set.
- **TargetTrack** - The same database that was used to retrieve the PSI: Biology and NESG datasets. Includes 287,844 soluble and 180,562 insoluble proteins based on experimental solubility values.
- **PDB** - The same database that was used to curate the Thera-SAbDab dataset. PDBSol used 402,059 soluble proteins from this database.
- **PRSP-2k** - A balanced database with 2,001 protein solubility instances.

After merging and removing duplicates, the initial dataset consisted of 198,164 soluble proteins and 113,471 insoluble proteins [83]. After pre-processing steps carried out in the original study,

such as removing redundancies and non-protein entities, the dataset consisted of 64,598 proteins. This dataset was then randomly split into a train (58,138 proteins), validation (3,230 proteins), and test set (3,230 proteins), which could be collected for this project.

The datasets, including their splits into train, validation, and test sets as well as solubility splits and expression system, are compiled in Figure 4.1.

Dataset	Train-Val-Test	Total	Ground Truth	Soluble-Insoluble	Expression
NESG	64%–16%–20%	1,323	Yes	63.6%–36.4%	E. Coli
PSI: Biology		11,226	Yes	66.8%–33.2%	E. Coli
eSOL		3,173	Yes*	44.1%–55.9%	in vitro
Superset		15,722	Yes	62.0%–38.0%	mixed
Thera-SAbDab		1,074**	No	N/A	N/A
PDBSol	90%–5%–5%	64,598	Yes	52.1%–47.9%	E. Coli

FIGURE 4.1: Summary of datasets used, including their data split (Train–Validation–Test), total sample count, availability of ground truth solubility labels, distribution of soluble vs. insoluble proteins, and the expression system. "E. Coli" refers to in vivo expression in Escherichia coli. *Originally continuous labels binarized using a 0.5 solubility threshold. **Each sample includes both heavy and light protein chains.

4.1.1 Pre-processing and Feature Extraction

The existing prediction models expected input data in FASTA format with a sequence identifier (SID) followed by a protein sequence as shown in Figure 3.2. Most datasets were retrieved in CSV format, where the amino acid sequence was provided in one of the columns alongside other metadata. In order to run the models, the data had to be converted to the FASTA format. This was achieved by retrieving the SID and protein sequence for each input data. In cases where ground truth labels were missing, the dataset contained duplicates, or incomplete entries, the file was processed in order to only include usable input.

The created prediction models used protein features as input instead of the amino acid sequences. These features were retrieved from the ProteinSol model, which produced 35 features for each protein sequence (see Section 3.3), as well as the computed SWI from the SWI model (see Section 3.2), resulting in 36 features for each protein sequence. The features are compiled in Figure 4.2 with corresponding feature numbers.

Feature No.	Feature
0	**K-R**
1	**D-E**
2	**Sequence length**
3	**A**
4	**C**
5	**D**
6	**E**
7	**F**
8	**G**
9	**H**
10	**I**
11	**K**
12	**L**
13	**M**
14	**N**
15	**P**
16	**Q**
17	**R**
18	**S**
19	**T**
20	**V**
21	**W**
22	**Y**
23	**K+R**
24	**D+E**
25	**K+R-D-E**
26	**K+R+D+E**
27	**F+W+Y**
28	**Isoelectric Point**
29	**Hydropathy**[a]
30	**Absolute Charge**[b]
31	**Fold Propensity**
32	**Disorder**
33	**Sequence Entropy**
34	**β-strand Propensity**
35	**SWI**

FIGURE 4.2: The extracted features with corresponding feature numbers. The letters correspond to amino acids and combinations of amino acids. SWI is extracted from the SWI model and the remaining features can be found in Section 2.2. (a - Kyte Doolittle scale, b - at pH 7)

4.2 Analyzing Datasets

Analyzing and comparing the datasets was essential for several reasons. NESG, eSOL, PSI: Biology, Superset, and PDBSol all consisted of single-protein sequences from E. Coli while Thera-SAbDab contained therapeutic antibody sequences, divided into heavy and light chains. Since the existing prediction models were trained on diverse protein sequences expressed in E. Coli, it was important to assess the degree of dissimilarity between Thera-SAbDab and the other datasets, as substantial differences could impact the model.

Additionally, understanding differences between the *E. Coli* datasets was crucial for evaluating model performance. For example, NetSolP was trained on an extended version of PSI: Biology but tested on NESG and eSOL, making it important to determine how similar these datasets were to the training set. Likewise, ProteinSol was based on the eSOL dataset, so comparing it to the other datasets helped assess potential biases or limitations in model generalization.

Furthermore, it is worth noting that the PDBSol dataset included protein sequences that were derived from the TargetTrack database, which also served as a source for some sequences in other datasets, such as NESG and PSI: Biology. Overlapping data sources may reduce dataset independence and inadvertently inflate performance metrics due to hidden data leakage. It was therefore valuable to assess how similar these datasets were in order to better interpret model generalization and avoid overestimating performance due to data leakage.

The dataset comparison was carried out by a logistic regression algorithm, trained to separate the datasets. In this work, all logistic regression models were created using the LogisticRegression class from Scikit-learn, with the solver chosen as *liblinear* (see [84]). The input data consisted of the normalized extracted protein features of two or more datasets with dataset labels added for each dataset. The input was then split into a training and testing set and the logistic regression model was trained on the training set in order to learn how to separate the input features into the corresponding dataset. Finally, the algorithm was used on the test set to predict which dataset the protein features belonged to. The results were collected in the form of a confusion matrix and accuracy score.

An important consideration during this analysis was the imbalance in dataset sizes. The datasets varied significantly in sample count, from 1,323 samples to 64,598 samples. This disparity could affect the logistic regression model's ability to generalize, potentially skewing towards the larger datasets. To mitigate this, downsampling was used in some cases in order to balance the larger datasets to match the size of the smaller ones being compared. This ensured fairer evaluation in how distinct the datasets were, without the results being dominated by the volume of data from any single source.

4.3 Applying Existing Prediction Models

To ensure reliability of the selected solubility models, their performances were validated by replicating benchmark results from a previous study. The models NetSolP, SWI, and ProteinSol were applied to the NESG dataset, following the same methodology outlined in article [70]. By comparing the evaluation metrics–accuracy, ROC-AUC, precision, recall, F_1-score, and Matthews Correlation Coefficient (MCC)–to those reported in the reference study, this step enabled verification that the models performed similarly, thereby allowing confirmation of their correct implementation.

To further assess agreement between models, their predictions were plotted against each other, incorporating solubility thresholds that yielded the highest model accuracy. A linear regression fit was included to visualize correlations. Additionally, a heatmap was generated to illustrate binary prediction agreement, showing the extent to which the models classified proteins as soluble or insoluble in similar manner. The same procedure was then applied to the eSOL dataset.

Finally, the models were applied to Thera-SAbDab, the dataset consisting of antibody protein sequences. Unlike previous datasets, Thera-SAbDab does not contain ground truth solubility labels, which precluded direct evaluation of model accuracy on this dataset. However, since the dataset included both heavy and light chain sequences, the models were applied separately to each type. This allowed for an independent assessment of solubility predictions for heavy and light chains, providing insight into potential differences in model behavior across different antibody protein sequences.

4.4 Creating Prediction Models

Several machine learning models were created in addition to the already existing models. The models were designed and implemented to try and improve predictive accuracy for solubility.

4.4.1 Machine Learning Models

The following models were based on more classic machine learning approaches such as logistic regression, SVMs, and clustering.

Feature Selection and Logistic Regression

The first model created was a simple logistic regression model that predicted solubility using the normalized features extracted from the data. The model was trained on the PSI: Biology dataset (11,226 samples) and tested on the NESG dataset (1,323 samples), with the aim of developing a model that generalizes well across datasets. L_1-regularization was applied to prevent overfitting and promote sparsity in the feature weights. The regularization strength, λ, was selected through grid search over the values $\lambda \in [0.001, 0.01, 0.1, 1, 10]$, based on performance during training.

Feature selection was applied to assess how different subsets of features influence solubility prediction performance (Figure 4.3). The number of selected features varied across $n \in [10, 15, 20, 25, 30, 35, 36]$, where $n = 36$ corresponds to using all available features. For each value of n, the top n features were selected based on different ranking methods. The feature selection techniques applied included χ^2 scoring, recursive feature elimination (RFE), random forest feature importance, and mutual information (MI).

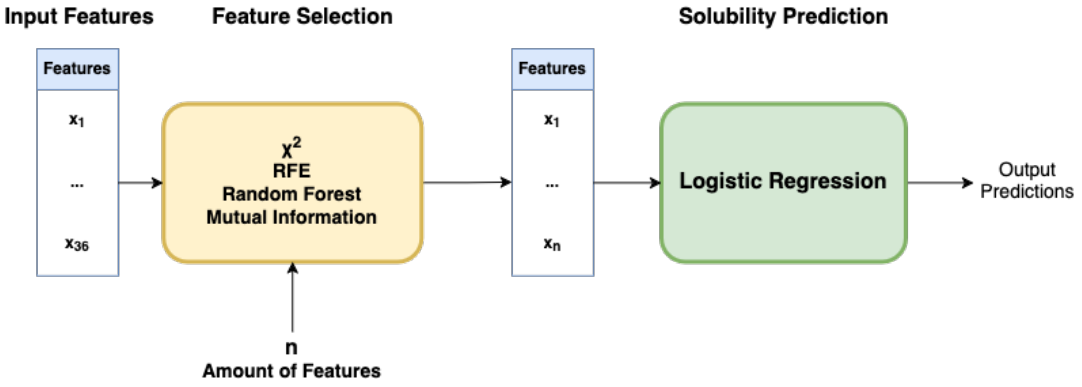


FIGURE 4.3: The pipeline of the model incorporating feature selection followed by logistic regression to predict solubility. Four different methods were used for feature selection and the amount of features, n, varied across $[10, 15, 20, 25, 30, 35, 36]$.

Only accuracy was used as the evaluation metric for this approach to keep the results interpretable and manageable, given the large number of feature selection methods, feature counts, and models tested. This model was intended as a rough baseline to explore whether feature selection had a large impact on model performance and whether generalization across datasets was feasible. As such, comprehensive metric reporting was not prioritized but would be relevant and appropriate in future work.

Feature Selection and Support Vector Machine

A model similar to the logistic regression approach described in above was developed, with the primary difference being the use of a Support Vector Machine (SVM) for solubility prediction (Figure 4.3). An RBF (Gaussian) kernel was employed to enable non-linear decision boundaries. As with the logistic regression model, a grid search was performed to tune the regularization parameter ($C \in 0.1, 1, 10$) and the kernel coefficient ($\gamma \in \text{scale}, \text{auto}$), which controls how much influence each training point has. The `scale` setting adjusts this based on both the number and spread of features, while `auto` considers only the number of features. Feature selection was also incorporated, using χ^2 scoring, recursive feature elimination (RFE), random forest feature importance, and mutual information (MI). The number of features considered was varied across $n \in [10, 15, 20, 25, 30, 35, 36]$, with 36 features representing the full feature set.

Logistic Regression

A logistic regression model was trained and evaluated using 5-fold cross-validation within a single dataset, repeated on NESG, eSOL, PSI: Biology, Superset, and PDBSol, with normalized features. This approach aimed to examine whether logistic regression could learn patterns related to solubility specific to that dataset. The model was also created in order to compare the model weights across different datasets to analyze if certain features contributed more than others in the predictions. L_1-regularization was once again employed, with the optimal value of λ selected through validation. The search was performed over 40 values spaced logarithmically between 10^{-3} and 10^{1}.

The averaged weights from the 5-folds of each logistic regression model trained on the different datasets were extracted and visualized. To further investigate similarities in feature importance across datasets, a correlation heatmap was generated based on the learned weight vectors. In addition, the most influential features were extracted. For each dataset, the top n features with the highest positive and negative weights were identified, as well as those whose weights were consistently close to zero (below a threshold of 10^{-3}). To quantify the consistency of feature importance across datasets, a score was computed by dividing the average absolute weight of each feature by its standard deviation across models. Features with the highest and lowest consistency scores were also reported, highlighting those with stable importance versus those whose relevance varied between datasets.

Logistic Regression for Dataset Prediction and Solubility Model Application

In this approach, a two-step model selection method was designed to test whether tailoring solubility predictions to dataset-specific models, after inferring dataset origin, could improve overall performance. A logistic regression model was trained to predict which dataset each test sample likely originated from–either NESG or eSOL-using the datasets' training splits. This model was developed using the same structure as in Section 4.4.1, but without applying 5-fold cross-validation individually to the logistic regression model.

Instead, 5-fold cross-validation was applied to the complete two-step procedure as a whole. That is, within each fold, a dataset prediction was first made, and then the corresponding solubility model was applied, ensuring that all evaluations were performed on unseen data (Figure 4.4). Once the dataset prediction was made, the corresponding existing solubility prediction model was applied. These solubility models had previously demonstrated the best performance for their respective datasets, as identified using the methodology in Section 4.3.

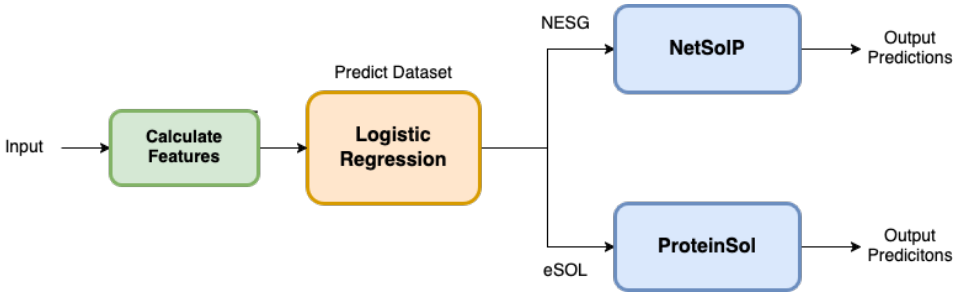


FIGURE 4.4: The pipeline of predicting dataset using logistic regression and then applying corresponding solubility model depending on predicted dataset.

To provide a meaningful comparison, new logistic regression models were also trained from scratch for each dataset using the training splits of the datasets. These models were then applied to the test samples assigned to their respective datasets, similarly to the existing models. This method followed a similar pipeline as shown in Figure 4.4, with the exception that the existing models, NetSolP and ProteinSol, instead were logistic regression models. This enabled an assessment of whether simple models trained on individual datasets could perform comparably to, or better than the existing models when using dataset prediction beforehand. As with the two-step procedure, 5-fold cross-validation was applied to this entire approach to ensure that all evaluations were performed on unseen data. This methodology was intended to test whether grouping similar samples, regardless of actual dataset label, and applying dataset-optimized models could improve performance.

Clustering and Logistic Regression

This model explores the idea of grouping proteins and training different logistic regression models on the groups of proteins with similar characteristics. The idea being that the models are specialized on specific protein groups, and therefore yield better predictive performance on such proteins.

All processing steps, including clustering, logistic regression model training, validation and testing, were performed using normalized features. The mean and standard deviation of each feature were stored for consistent normalization of the test data.

Protein grouping was performed using K-means clustering [85] on all of the normalized features of the proteins in the training and validation sets. The proteins in each of the clusters were divided into train and validation sets. Subsequently, a logistic regression model [86] was trained on each individual cluster, using the validation sets as before to find the optimal value of λ for L_1-regularization.

Each protein in the test set was assigned to a cluster using one of three classifiers: K-Nearest Neighbors (KNN) (see Section 2.3.1), Nearest centroid (see Section 2.3.1) and Radian Basis Function (RBF) (see Section 2.3.4) [87]. Once assigned, each protein was evaluated using the logistic regression model corresponding to its cluster. The process of the model is shown as a flowchart in Figure 4.5. This model was evaluated using 5-fold cross-validation where the whole process was recomputed for each fold. However, in practical use, the clusters and trained models were saved, so that when new proteins were evaluated, only feature calculations, cluster classification and logistic regression models needed to be applied.

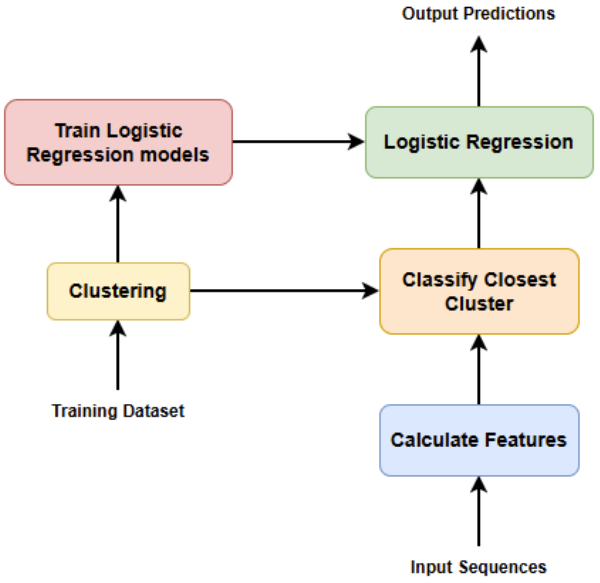


FIGURE 4.5: The flowchart of the clustering-based logistic regression model.

With this model, two different experiments were performed, the first being the effect of the number of clusters on the result. The model was also evaluated for training on all the train sets of the NESG, eSOL, PSI: Biology, Superset and PDBSol datasets and then tested on the datasets' corresponding test sets.

4.4.2 Artificial Neural Networks

As an addition to the more traditional machine learning models, various multi-layer perceptron models (MLPs) were created with the aim of capturing more complex patterns in the data.

Multi-Layer Perceptron

Three main MLPs were created using the *TensorFlow* library in *Python*. The models differed in their size and complexity:

- **Model 1**: 2-4-1 architecture (three layers with two, four, and one neurons respectively (Figure 4.6))
- **Model 2**: 4-8-1 architecture
- **Model 3**: 256-128-64-1 architecture

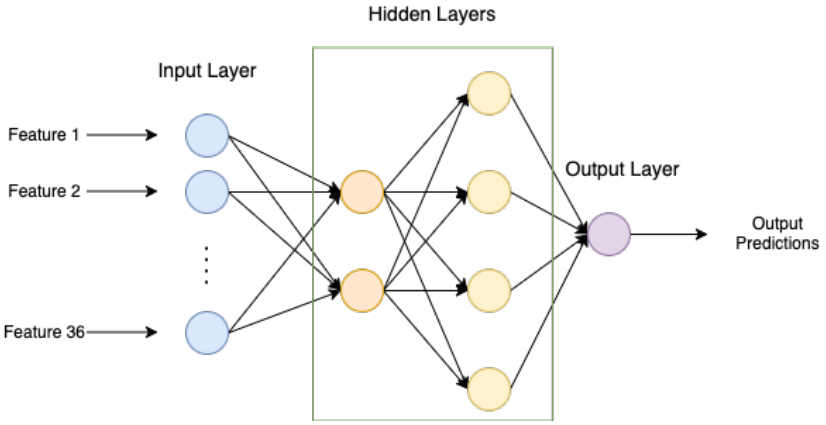


FIGURE 4.6: A flowchart of a MLP with a 2-4-1 structure. The input is passed to a hidden layer with two neurons, followed by another hidden layer with four neurons until it reaches the single output neuron.

In all models, the output layer consisted of a single neuron with a sigmoid activation function to produce a solubility probability score. The hidden layers used leaky-ReLU as activation functions to better capture complex patterns in the data and avoid "dying neurons". All models followed a similar structure as seen in Figure 4.6 but had varying amounts of neurons and hidden layers. Similar to previous models, 5-fold cross-validation was applied to the models.

To prevent overfitting, each model incorporated the following regularization and training strategies. The parameters used were initially based on commonly accepted default values and were subsequently fine-tuned based on validation performance and practical considerations during development:

- **L_2-regularization** with a coefficient of 0.001 on the layers' weights.
- **Dropout and Batch Normalization** between layers. The dropout was set to 0.3, meaning that it dropped 30% of the neurons randomly during training.
- **ADAM optimizer** with an initial learning rate of 0.001.
- **Binary Cross-Entropy** as the loss function.
- **Early Stopping** with a patience of 30 epochs, within a maximum of 300 training epochs.
- **Learning Rate Decay** on plateau by a factor of 0.8 and a patience of 15 epochs.

MLP Comparison with Logistic Regression

To directly compare with logistic regression, a simple perceptron was implemented using a single neuron with a sigmoid activation function, effectively replicating the behavior of logistic

regression in a neural network framework.

In a second experiment, a minimal MLP with a 2-1 architecture was constructed as shown in Figure 4.7. The hidden layer contained two neurons where one neuron was initialized with the exact weights from a trained logistic regression model. The other neuron had its weights frozen at zero, effectively contributing nothing to the output. The output layer then applied a sigmoid activation function as before.

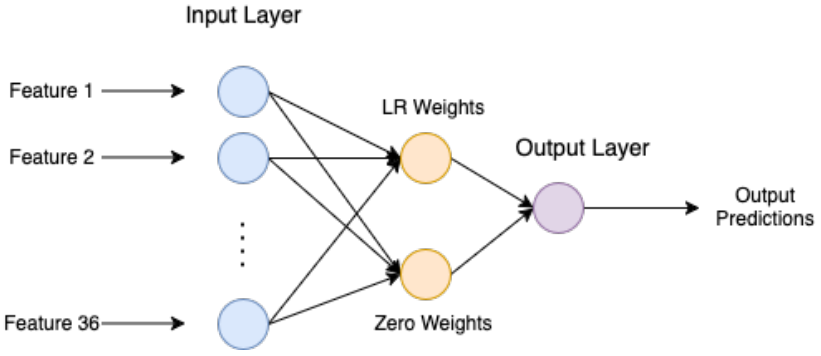


FIGURE 4.7: A flowchart of a MLP using a 2-1 structure where one neuron has logistic regression model weights while the other one has weights set to zero.

This setup was further modified to explore whether the model could learn patterns beyond those captured by logistic regression. Specifically, the previously frozen zero-weight neuron was made trainable, allowing it to adjust its weights during training. This enabled the model to potentially capture additional nonlinear patterns while still incorporating the baseline behavior of logistic regression.

For all of these cases, 5-fold cross-validation was applied across the entire procedure to ensure that evaluation was performed consistently and fairly across different splits of the data.

4.5 Tools and Resources

Throughout the development of this thesis, the following tools and resources were utilized:

- **Laptops**: Two laptops provided by collaborating company with MacOS and Windows operating systems respectively.
- **Python**: The primary programming language used for data processing, model development, and evaluation.

- **Libraries**: Key libraries included `NumPy` and `pandas` for data manipulation, `matplotlib` and `seaborn` for data visualization, `scikit-learn` for machine learning, and `TensorFlow` for artificial neural networks.
- **PyCharm Community Edition and Visual Studio Code**: The integrated development environments (IDEs) for code development.
- **Git**: Utilized for version control to track changes and manage the coding process.
- **ChatGPT**: Utilized as a tool for refining language, ensuring grammatical accuracy, and assisting in troubleshooting coding challenges.

CHAPTER 5

Results

5.1 Dataset Analysis

The Figures 5.1, 5.2, 5.3, 5.4, and 5.5 display confusion matrices from logistic regression models trained to predict dataset origin from extracted features.

Figure 5.1 shows dataset predictions for NESG, eSOL, and Thera-SAbDab heavy and light chains with an overall accuracy of 0.84. Most misclassifications occurred between the NESG and eSOL datasets while the Thera-SAbDab heavy and light chain proteins were only misclassified once each.

		Predicted			
		NESG	**eSOL**	**Heavy***	**Light****
Actual	**NESG**	81	195	0	1
	eSOL	43	803	8	4
	Heavy*	0	0	207	1
	Light**	0	1	0	206

FIGURE 5.1: Shows the confusion matrix for logistic regression of NESG, eSOL, *Thera-SAbDab heavy chain, and **Thera-SAbDab light chain features. Accuracy: 0.84.

Figure 5.2 shows the confusion matrix for NESG and PSI: Biology predictions. The PSI: Biology dataset (11,226 samples) was downsampled to match NESG (1,323 samples) yielding a prediction

accuracy of 0.52 and an almost even amount of true positives, false positives, true negatives, and false negatives. Figure 5.3 shows the results for eSOL (3,173 samples) and PSI: Biology, where the latter was again downsampled, resulting in an accuracy of 0.72.

		Predicted	
		NESG	**PSI**
Actual	**NESG**	136	132
	PSI	125	137

FIGURE 5.2: Shows the confusion matrix for logistic regression of NESG and PSI: Biology features. The PSI: Biology entries were downsampled to match NESG. Accuracy: 0.52.

		Predicted	
		eSOL	**PSI**
Actual	**eSOL**	484	177
	PSI	175	433

FIGURE 5.3: Shows the confusion matrix for logistic regression of eSOL and PSI: Biology features. The PSI: Biology entries were downsampled to match eSOL. Accuracy: 0.72.

Figure 5.4 shows the confusion matrix for predictions across NESG, eSOL, PSI: Biology, and PDBSol (64,598 samples). All datasets were downsampled to match NESG, resulting in an accuracy of 0.43.

		Predicted			
		NESG	**eSOL**	**PSI**	**PDBSol**
Actual	**NESG**	70	76	71	61
	eSOL	21	182	37	24
	PSI	45	64	72	56
	PDBSol	47	52	49	131

FIGURE 5.4: Shows the confusion matrix for logistic regression of NESG, eSOL, PSI: Biology, and PDBSol with downsampled datasets to match NESG. Accuracy: 0.43.

Lastly, Figure 5.5 shows the confusion matrix for PSI: Biology and PDBSol. PDBSol was downsampled to match PSI: Biology and the resulting accuracy was 0.66.

		Predicted	
		PSI	**PDBSol**
Actual	**PSI**	1602	655
	PDBSol	876	1358

FIGURE 5.5: Shows the confusion matrix for logistic regression of PSI: Biology and PDBSol features. The PDBSol entries were downsampled to match PSI: Biology. Accuracy: 0.66.

5.2 Existing Models

The following sections showcase the results from applying the existing models NetSolP, SWI, and ProteinSol for solubility prediction on the NESG, eSOL, and Thera-SAbDab datasets.

5.2.1 NESG Dataset

Figure 5.6 shows the solubility prediction results on the NESG dataset using the existing models NetSolP, SWI, and ProteinSol. The optimal accuracy thresholds for each model were 0.68 (NetSolP), 0.31 (ProteinSol), and 0.50 (SWI).

Method	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NetSolP^a	**0.730**	**0.760**	**0.773**	0.817	**0.794**	**0.407**
ProteinSol^b	0.658	0.679	0.664	**0.933**	0.776	0.169
SWI^c	0.680	0.690	0.712	0.895	0.783	0.270

FIGURE 5.6: Shows accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient (MCC) on the NESG dataset with thresholds 0.68^a, 0.31^b, and 0.50^c, resulting in the highest accuracies. The bold values represent the highest metrics. The dataset consisted of 1,323 protein sequences.

Figure 5.7 shows pairwise scatterplots of solubility predictions on the NESG dataset, with optimal accuracy thresholds and linear regression fitted lines. Points right of a vertical threshold are predicted as soluble by the x-axis model while points above a horizontal threshold are soluble per the y-axis model. Figure 5.8 complements this with a heatmap showing binary prediction agreement between the models, representing the percentage of samples predicted as either soluble or insoluble by both. These correspond to the upper-right and lower-left quadrants of the scatterplots divided by the model thresholds. The binary prediction agreement was similar for ProteinSol compared to SWI (74.38%) and NetSolP (73.02%) while the agreement between SWI and NetSolP yielded a slightly lower percentage (67.65%).

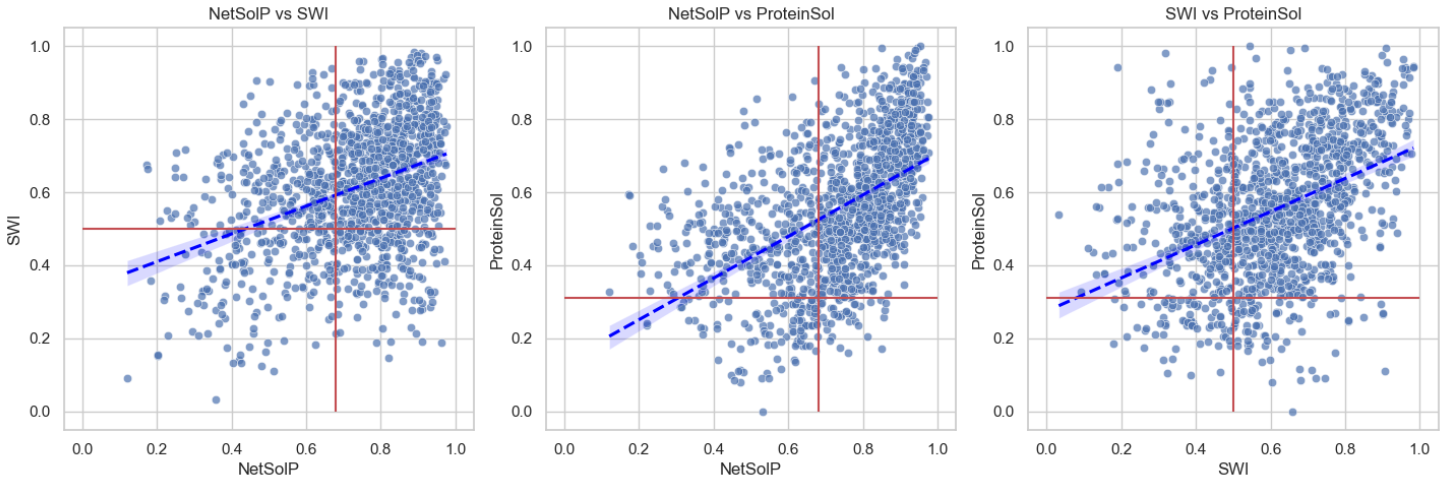


FIGURE 5.7: Shows the predicted protein solubility for the NESG dataset in pairwise scatter plots. The models' corresponding thresholds (red lines) resulting in highest accuracies are added as well as a linear regression fit (blue dashed line).

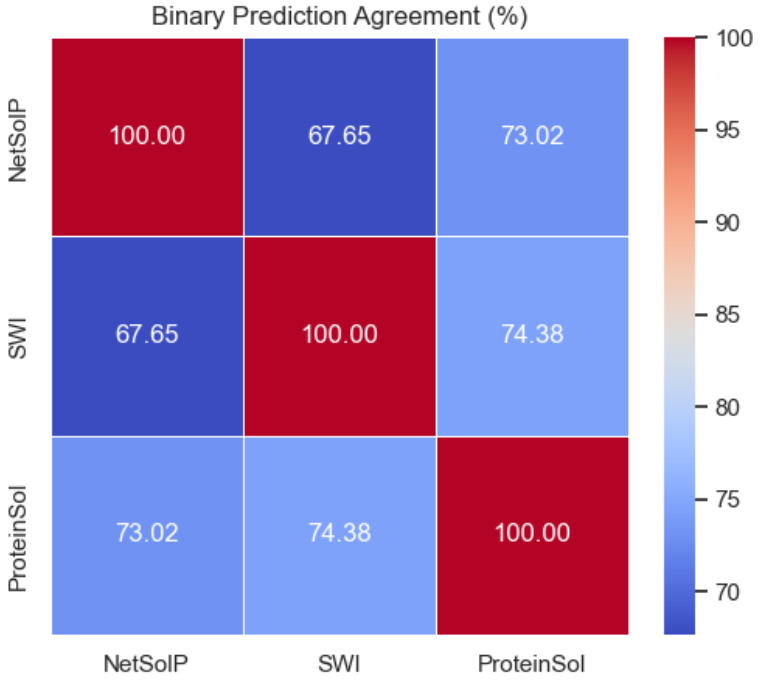


FIGURE 5.8: Shows a heatmap of binary prediction agreement between the models on the NESG dataset.

5.2.2 eSOL Dataset

Figure 5.9 shows solubility prediction results on the eSOL dataset, using NetSolP, SWI, and ProteinSol. The optimal accuracy thresholds were 0.81 (NetSolP), 0.51 (ProteinSol), and 0.63

(SWI).

Method	ACC	AUC	PRE	REC	F1	MCC
NetSolP[a]	0.692	0.733	0.699	0.548	0.614	0.372
ProteinSol[b]	**0.732**	**0.795**	**0.742**	**0.615**	**0.672**	**0.454**
SWI[c]	0.698	0.748	0.690	0.592	0.637	0.385

FIGURE 5.9: Shows accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient (MCC) on the eSOL dataset with thresholds 0.81[a], 0.51[b], and 0.63[c], resulting in the highest accuracies. The bold values represent the highest metrics. The dataset consisted of 3,173 protein sequences.

Similar to the NESG results, Figures 5.10 and 5.11 show the pairwise scatterplots and binary prediction agreement heatmap for solubility predictions on the eSOL dataset. The binary prediction agreement varied slightly more between the models (between 55.95% to 74.55%) compared to their agreement on the NESG dataset.

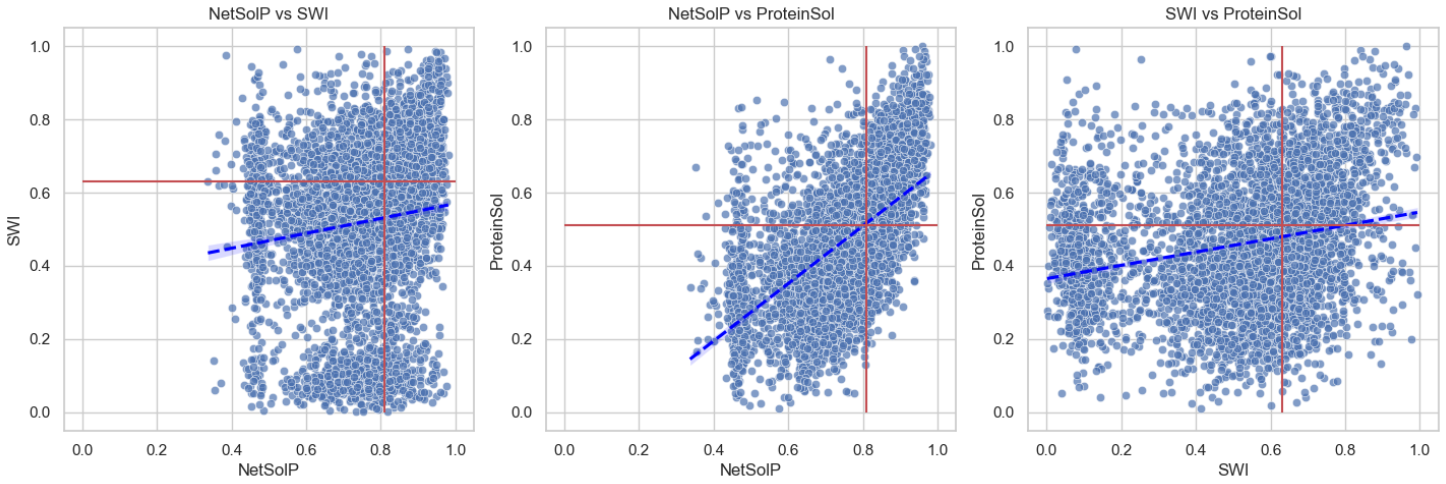


FIGURE 5.10: Shows the predicted protein solubility for the eSOL dataset in pairwise scatter plots. The models' corresponding thresholds (red lines) resulting in the highest accuracies are added as well as a linear regression fit (blue dashed line).

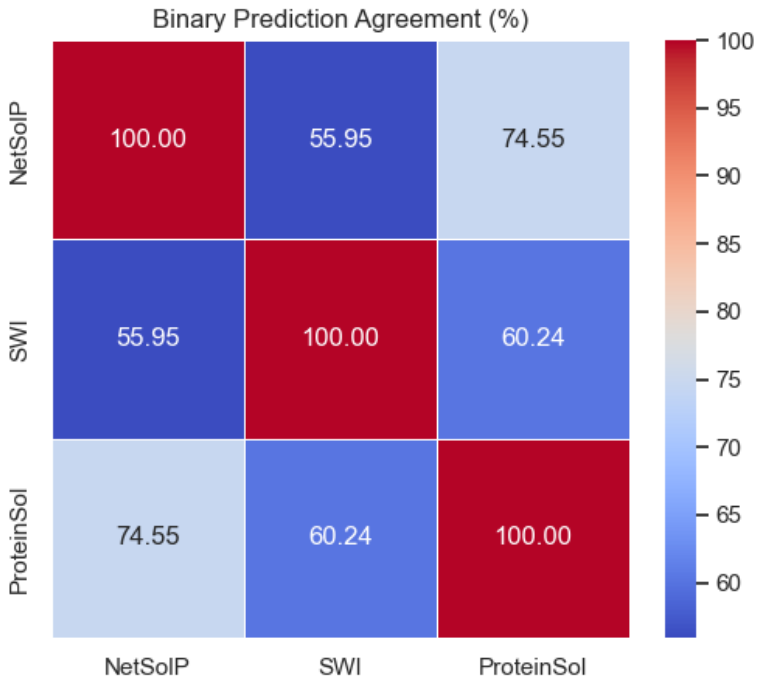


FIGURE 5.11: Shows a heatmap of binary prediction agreement between the models on the eSOL dataset.

5.2.3 Thera-SAbDab Dataset

Figure 5.12 shows pairwise scatterplots of solubility predictions on Thera-SAbDab heavy chain samples. Since ground truth was unavailable, thresholds from the NESG and eSOL results are used for binary prediction. Figure 5.13 presents the corresponding binary prediction agreement heatmaps for each threshold set. In this case, the binary prediction agreement varies a lot and is either very low (3.08% and 38.34%) or very high (83.30% and 99.72%) with no in between ranges.

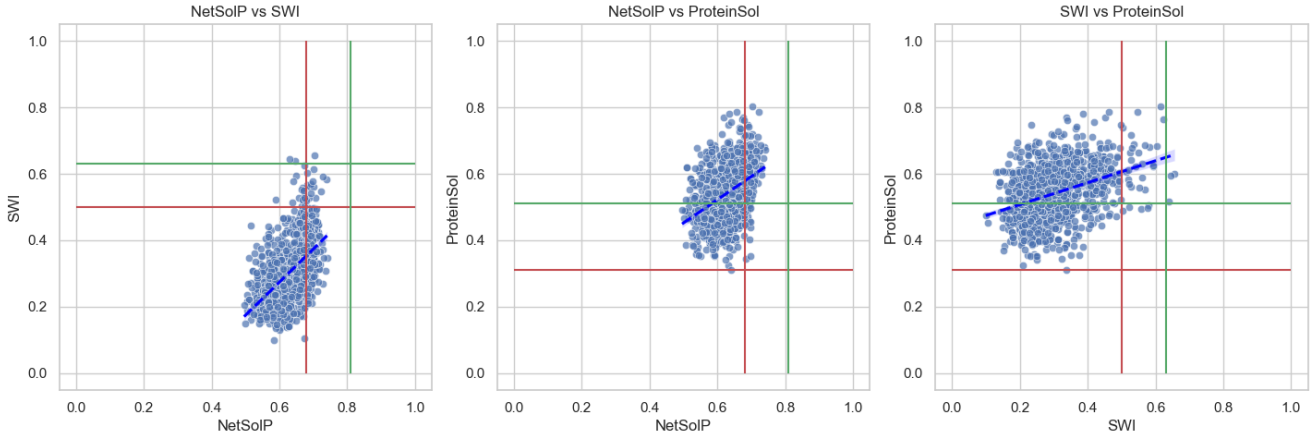


FIGURE 5.12: Shows the predicted protein solubility for the Thera-SAbDab dataset with heavy chains in pairwise scatter plots. The models' corresponding thresholds resulting in the highest accuracies for the NESG (red threshold) and eSOL (green threshold) datasets are added as well as a linear regression fit (blue dashed line). Dataset consisted of 1,074 protein sequences.

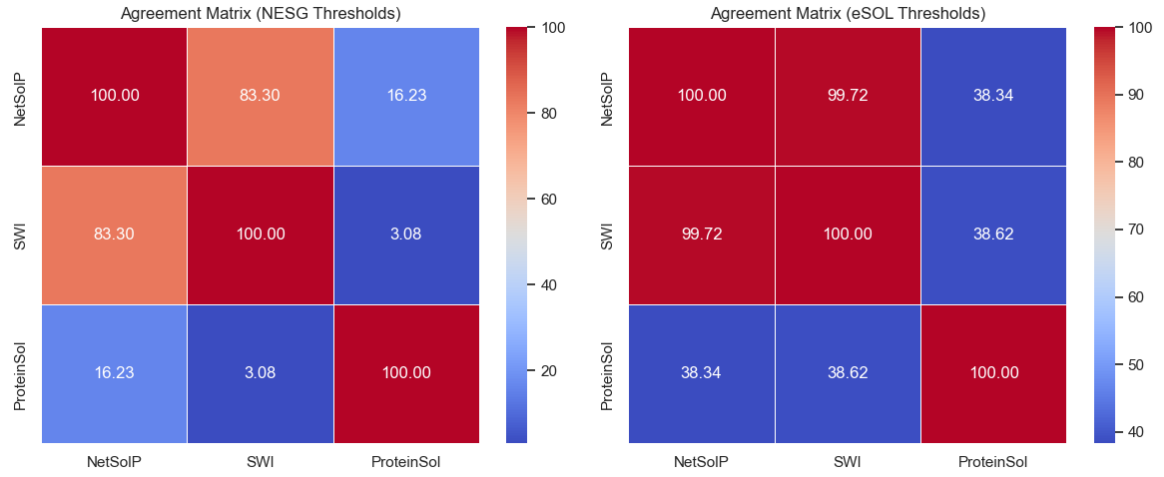


FIGURE 5.13: Shows heatmaps of binary prediction agreement (in precent) between the models on the Thera-SAbDab dataset with heavy chains using the thresholds that result in highest accuracies for the NESG (left) and eSOL (right) datasets.

Similarly, Figures 5.14 and 5.15 show scatterplots and binary prediction agreement heatmaps for solubility predictions on the Thera-SAbDab light chain samples. Similarly to above, the binary prediction agreements are on two extremes (7.25% and 25.95% or 89.79% and 100.00%).

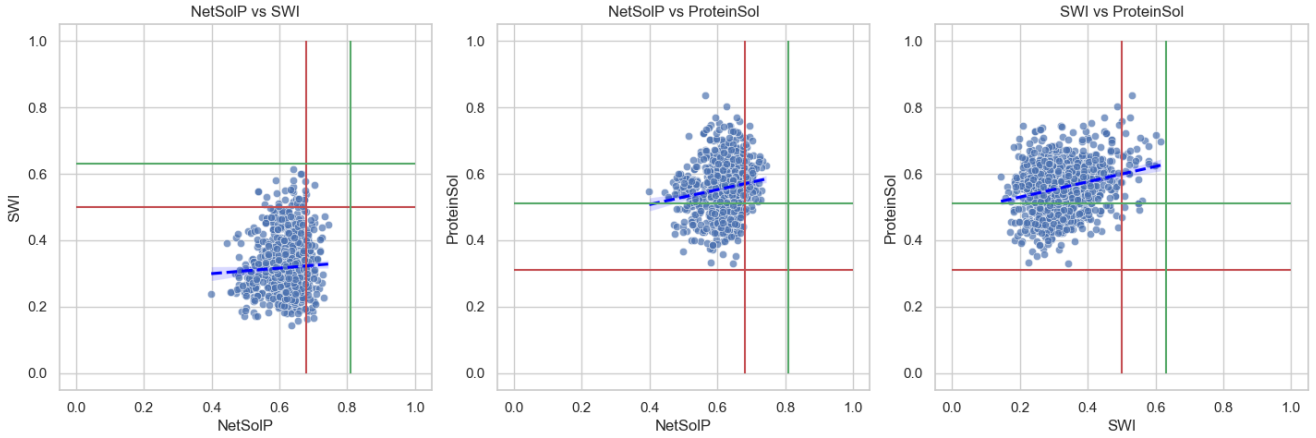


FIGURE 5.14: Shows the predicted protein solubility for the Thera-SAbDab dataset with light chains in pairwise scatter plots. The models' corresponding thresholds resulting in the highest accuracies for the NESG (red threshold) and eSOL (green threshold) datasets are added as well as a linear regression fit. Dataset consisted of 1,074 protein sequences.

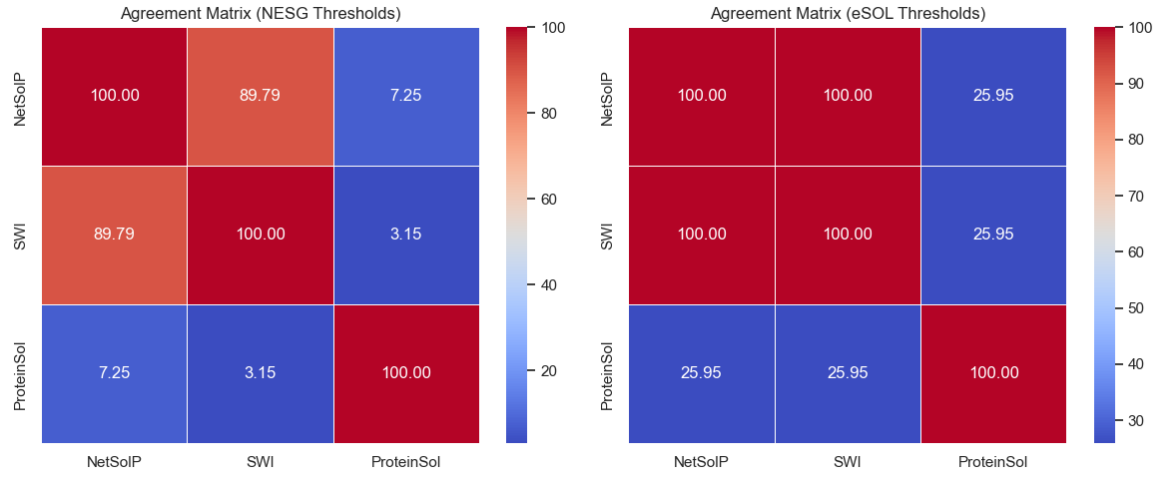


FIGURE 5.15: Shows a heatmap of binary prediction agreement (in percent) between the models on the Thera-SAbDab dataset with light chains using the thresholds that result in highest accuracies for the NESG (left) and eSOL (right) datasets.

5.3 Created Models

This section presents the results from applying the created prediction models to the NESG, eSOL, PSI: Biology, Superset, and PDBSol datasets.

5.3.1 Machine Learning Models

The following sections highlight the results from models developed using traditional machine learning approaches.

Feature Selection and Machine Learning

Figure 5.16 shows the accuracy of logistic regression and support vector machine (SVM) models with feature selection, trained on the PSI: Biology dataset and tested on the NESG dataset. The bold values indicate the highest accuracy for each model: 0.695 for logistic regression with 15 features selected via random forest, and 0.702 for SVM with 15 features selected through recursive feature elimination. Results using χ^2 and *Mutual Information* for feature selection are shown in Figure A.3, and the chosen grid search parameters are in Figure A.1.

		Prediction Model	
Feature Selection	**Number of Features**	**Logistic Regression**	**SVM**
Recursive Feature Elimination	**10**	0.689	0.697
	15	0.691	**0.702**
	20	0.689	0.696
	25	0.690	0.688
	30	0.692	0.685
	35	0.691	0.688
Random Forest	**10**	0.684	0.678
	15	**0.695**	0.686
	20	0.694	0.684
	25	0.691	0.685
	30	0.691	0.687
	35	0.691	0.682
All Features	**36**	0.690	0.688

FIGURE 5.16: Shows best accuracy for different combinations of prediction models and feature selection on NESG dataset with 1,323 sequences. Model is trained on PSI: Biology dataset with 11,226 sequences. The bold results indicate the highest accuracy for each model type. Results using the feature selection methods χ^2 and *Mutual Information* can be found in Figure A.3. Exact parameters for models are found in Figure A.1.

Feature Selection	Number of Features	Accuracy Logistic Regression
Recursive Feature Elimination	**10**	0.689
	15	0.691
	20	0.689
	25	0.690
	30	0.692
	35	0.691
Random Forest	**10**	0.684
	15	**0.695**
	20	0.694
	25	0.691
	30	0.691
	35	0.691
All Features	**36**	0.690

FIGURE 5.17: Accuracy of logistic regression with different feature selection methods and number of features.

Logistic Regression

Figure 5.18 shows the results from logistic regression models trained and tested on the NESG, eSOL, PSI: Biology, Superset, and PDBSol datasets. The model trained on eSOL achieved the highest accuracy (0.7803), AUC (0.8452), precision (0.7783), and MCC (0.5532). The accuracy for the NESG dataset (0.7080) was also higher than all trials in the above method using feature selection.

	Metric					
Dataset	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NESG	0.7080	0.7101	0.7480	0.8227	0.7809	0.3557
eSOL	**0.7803**	**0.8452**	**0.7783**	0.7048	0.7385	**0.5532**
PSI	0.7065	0.7292	0.7407	**0.8633**	**0.7972**	0.2882
Superset	0.7076	0.7570	0.7431	0.8080	0.7738	0.3656
PDBSol	0.6162	0.6593	0.6058	0.7558	0.6720	0.2314

FIGURE 5.18: Shows the average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient (MCC) using the logistic regression model with 5-fold cross-validation and best λ-value for L_1-regularization from validation set at each run. The bold values represent the highest metrics.

Figure 5.19 shows the correlation between the feature weights of each logistic regression model. The model trained on PDBSol achieved the lowest feature weight correlation to the remaining dataset models (0.14 to 0.31). The correlation heatmap is supported by Figure 5.20, which plots the feature weights. The feature indices on the x-axis match the numbers in Figure 4.2. Detailed information on which feature weights were positive, negative, or zero for each model is presented in Figure A.2.

FIGURE 5.19: Shows a heatmap of the correlation between feature weights in the logistic regression models for different datasets. Details of feature weights are found in Figure A.2.

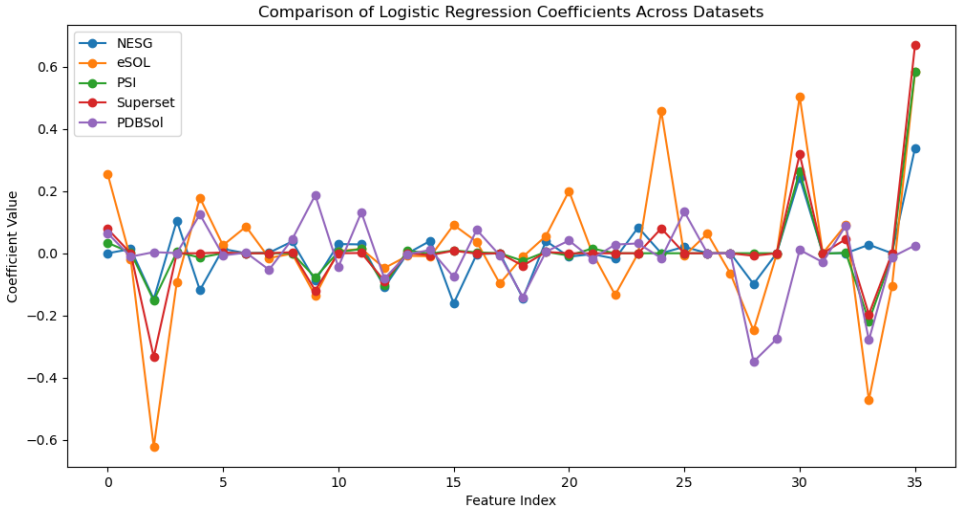


FIGURE 5.20: Shows the averaged weights for the logistic regression models trained on different datasets with 5-fold cross-validation. Details of feature weights are found in Figure A.2.

Logistic Regression for Dataset Prediction and Solubility Model Application

Figure 5.21 shows the results from first using logistic regression to predict which dataset a sample belongs to, followed by solubility prediction using the best performing existing model and logistic regression for comparison. NetSolP was applied to samples predicted as NESG, and ProteinSol was applied to samples predicted as eSOL, as they performed with highest accuracy as shown in Figures 5.6 and 5.9.

		Metric					
Model	**Predicted Dataset**	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NetSolP	**NESG**	0.7159	0.7716	0.7327	0.8879	0.8028	0.3296
ProteinSol	**eSOL**	0.7026	0.7452	0.6761	0.4923	0.5687	0.3611
Logistic Regression	**NESG**	0.7073	0.6979	0.8037	0.7294	0.7644	0.3838
	eSOL	0.7426	0.7251	0.6945	0.6385	0.6645	0.4582

FIGURE 5.21: Shows average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coeffcient (MCC) from 5-fold cross-validation with the model using logistic regression to predict dataset and then applying corresponding best performing existing model as well as a logistic regression model.

Clustering and Logistic Regression

As shown in Figure 5.22, the performance differences between the cluster classifiers are relatively small. However, the RBF classifier consistently achieved slightly higher accuracy and AUC compared to the other classifiers. The same results were observed for the other datasets, as

shown in Appendix A.4. Based on this observation, the RBF classifier was selected for use in subsequent tests.

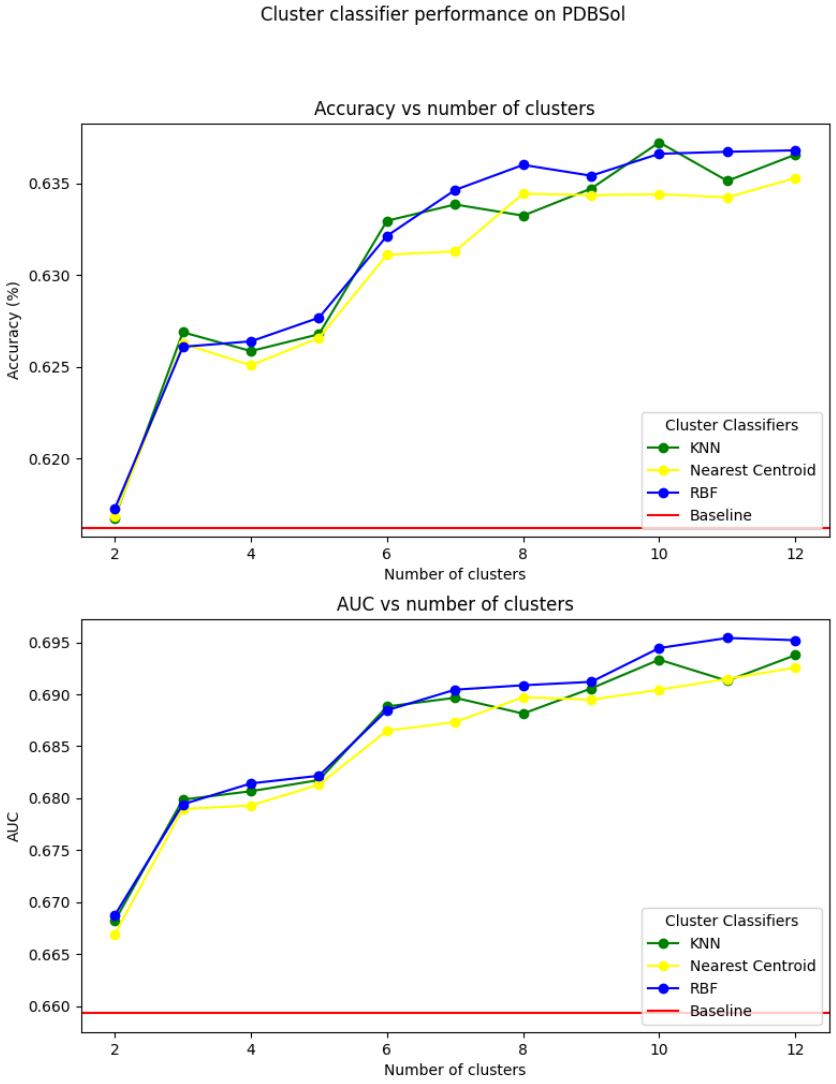


FIGURE 5.22: Shows the average accuracy and area under ROC curve (AUC) against the number of clusters over 5-fold cross-validation of the clustering and logistic regression model. KNN, nearest centroid and RBF was used as cluster classifiers. The PDBSol's training and validation sets were used for clustering and training, while its test set was used for evaluation. The baseline is the value from the logistic regression model on PDBSol.

Figure 5.23 represents the accuracy and area under the ROC curve (AUC) from the clustering-based logistic regression model predicting solubility, plotted against the number of clusters. The models were trained and tested on the NESG, eSOL, PSI, Superset and PDBSol datasets. The figure shows that the use of more clusters resulted in a smaller accuracy and AUC for NESG and eSOL. The performance on PSI and the Superset remained relatively unaffected by the number of clusters, but on PDBSol, the performance increased with the number of clusters. Overall,

the model performance is relatively stable across different number of clusters, while the dataset has a more noticeable effect on the metrics.

The number of clusters that achieved the best performance for each dataset was used in the final evaluation of this model. Specifically, the optimal number of clusters (from Figure 5.23) was 2 for NESG, 3 for eSOL, 5 for PSI: Biology, 4 for the Superset and 11 for PDBSol. Figure 5.24 shows all the evaluation metrics obtained using these number of clusters on the five datasets.

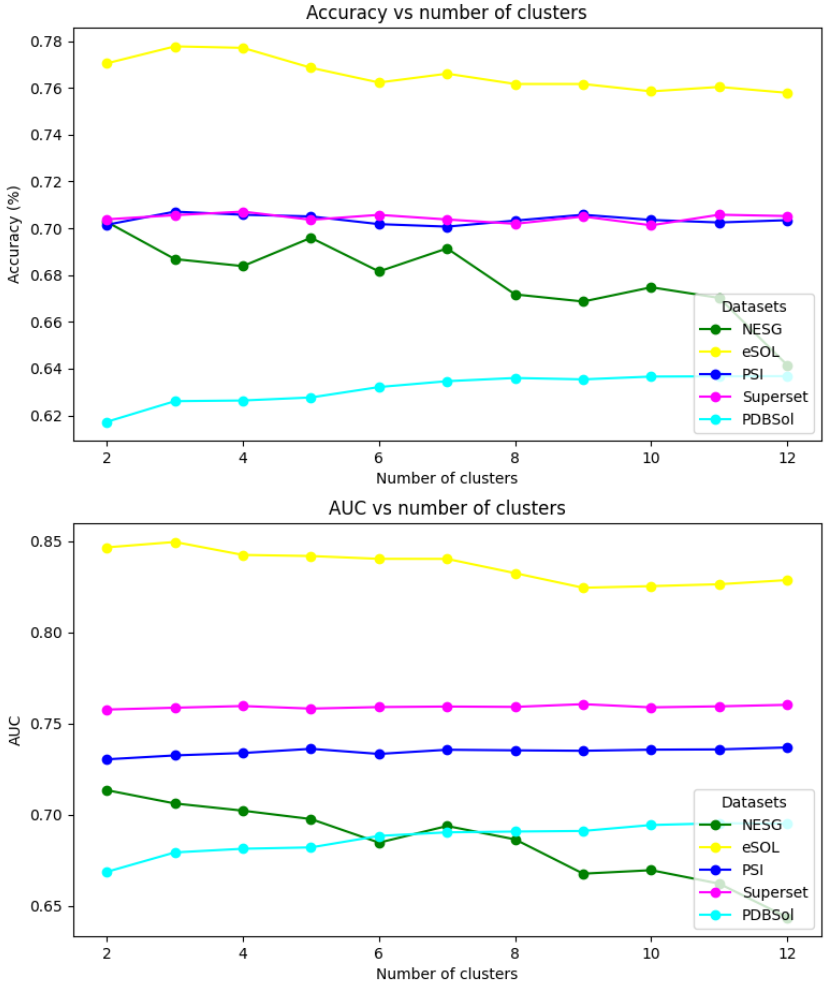


FIGURE 5.23: Shows the average accuracy and area under ROC curve (AUC) against the number of clusters over 5-fold cross-validation of the clustering and logistic regression model. RBF was used as a cluster classifier. Clustering and training was done on train and validation sets and evaluated on respective test sets.

	Metric					
Dataset	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NESG	0.7027	0.7136	0.7236	**0.8632**	0.7869	0.3216
eSOL	**0.7777**	**0.8497**	**0.7698**	0.7091	0.7380	**0.5472**
PSI	0.7051	0.7362	0.7401	0.8609	**0.7959**	0.2855
Superset	0.7071	0.7597	0.7362	0.8219	0.7766	0.3597
PDBSol	0.6367	0.6954	0.6367	0.7047	0.6689	0.2705

FIGURE 5.24: Shows the average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient (MCC) over 5-fold cross-validation of the clustering and logistic regression model. The number of clusters used for each dataset corresponds to the configuration that yielded the best performance: 2 for NESG, 3 for eSOL, 5 for PSI: Biology, 4 for the Superset and 11 for PDBSol.

5.3.2 Artificial Neural Networks

This section presents the results from applying the created MLPs to the NESG, eSOL, PSI: Biology, Superset, and PDBSol datasets.

Multi-Layer Perceptron

Figure 5.25 shows solubility prediction results for the NESG, eSOL, PSI: Biology, Superset, and PDBSol datasets using three multi-layer perceptrons (MLPs) with architectures 2-4-1 (MLP 1), 4-8-1 (MLP 2), and 256-128-64-1 (MLP 3). All datasets achieved higher accuracy with MLP 3 compared to MLP 1, with the lowest increase being 0.0165 (PSI: Biology) and the highest being 0.0252 (PDBSol).

Model	Dataset	Metric					
		ACC	AUC	PRE	REC	F1	MCC
1	**NESG**	0.6581	0.6806	0.6575	0.9687	0.7827	0.1282
	eSOL	**0.7654**	**0.8294**	**0.7695**	0.6717	0.7168	**0.5220**
	PSI	0.6748	0.7139	0.6782	**0.9765**	**0.8004**	0.1048
	Superset	0.6620	0.7432	0.6617	0.9309	0.7733	0.2237
	PDBSol	0.6104	0.6523	0.6012	0.7495	0.6669	0.2190
2	**NESG**	0.6490	0.6880	0.6524	0.9596	0.7762	0.1320
	eSOL	**0.7708**	**0.8367**	**0.7636**	0.6981	0.7288	**0.5328**
	PSI	0.6749	0.7197	0.6815	**0.9657**	**0.7987**	0.1148
	Superset	0.6781	0.7463	0.6806	0.9054	0.7770	0.2719
	PDBSol	0.6183	0.6670	0.6048	0.7742	0.6784	0.2376
3	**NESG**	0.6755	0.7009	0.6815	0.9157	0.7814	0.2366
	eSOL	**0.7878**	**0.8559**	**0.7582**	0.7663	0.7616	**0.5711**
	PSI	0.6913	0.7328	0.7007	**0.9394**	**0.8025**	0.2014
	Superset	0.6824	0.7513	0.6872	0.8973	0.7779	0.2851
	PDBSol	0.6356	0.6929	0.6138	0.8116	0.6987	0.2765

FIGURE 5.25: Shows the average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient (MCC) over 5-fold cross-validation of the three main MLPs created. The models have the architectures 2-4-1, 4-8-1, and 256-128-64-1 respectively.

Multi-Layer Perceptrons Mimicking Logistic Regression

Figure 5.26 shows solubility prediction results on the NESG, eSOL, PSI: Biology, Superset, and PDBSol datasets using a single sigmoid activation layer to mimic logistic regression.

Dataset	Metric					
	ACC	AUC	PRE	REC	F1	MCC
NESG	0.6786	0.7050	0.6843	0.9191	0.7842	0.2389
eSOL	**0.7724**	**0.8349**	**0.7544**	0.7195	0.7361	**0.5370**
PSI	0.6848	0.7197	0.6974	**0.9333**	**0.7982**	0.1797
Superset	0.6696	0.7457	0.6700	0.9205	0.7754	0.2474
PDBSol	0.6118	0.6564	0.6177	0.6685	0.6420	0.2203

FIGURE 5.26: Shows the average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient from 5-fold cross-validation of a MLP using only the sigmoid output.

Figure 5.27 shows solubility prediction results using a 2-1 MLP, where one neuron has weights from a logistic regression model and the other has weights frozen at zero in the hidden layer. Similar results are shown in Figure 5.28, where the second neuron has trainable weights. For reference, the results from the logistic regression model, from which the weights were sourced, are in Figure 5.18.

	Metric					
Dataset	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NESG	0.6891	0.7141	0.7055	0.8811	0.7824	0.2858
eSOL	**0.7708**	**0.8440**	**0.7715**	0.6826	0.7241	**0.5326**
PSI	0.6904	0.7265	0.7042	**0.9269**	**0.7998**	0.2059
Superset	0.7004	0.7514	0.7246	0.8338	0.7752	0.3404
PDBSol	0.6101	0.6594	0.6204	0.6477	0.6338	0.2176

FIGURE 5.27: Shows the average accuracy, area under ROC curve (AUC), precision, recall, F_1-score, and Matthew's Correlation Coefficient from 5-fold cross-validation of a 2-1 MLP using frozen logistic regression weights for one neuron and frozen zero weights for the other neuron followed by a sigmoid activation function for the output. The performance for the logistic regression model from which the weights were taken from can be found in Figure 5.18.

	Metric					
Dataset	**ACC**	**AUC**	**PRE**	**REC**	**F1**	**MCC**
NESG	0.6725	0.7082	0.7167	0.8026	0.7570	0.2635
eSOL	**0.7718**	**0.8433**	**0.7729**	0.6841	0.7254	**0.5346**
PSI	0.7014	0.7267	0.7370	**0.8601**	**0.7938**	0.2749
Superset	0.7048	0.7554	0.7465	0.7928	0.7689	0.3626
PDBSol	0.6103	0.6592	0.6203	0.6496	0.6345	0.2179

FIGURE 5.28: Shows the average accuracy, area under ROC curve, precision, recall, F_1-score, and Matthew's Correlation Coefficient from 5-fold cross-validation of a 2-1 MLP using frozen logistic regression weights for one neuron and trainable weights for the other neuron followed by a sigmoid activation function for the output. The performance for the logistic regression model from which the weights were taken from can be found in Figure 5.18.

5.3.3 Model Comparison

Figures 5.29, 5.30, 5.31, 5.32, and 5.33 show plots of the metrics normalized over each metric across all datasets for models applied to NESG, eSOL, PSI: Biology, Superset, and PDBSol respectively.

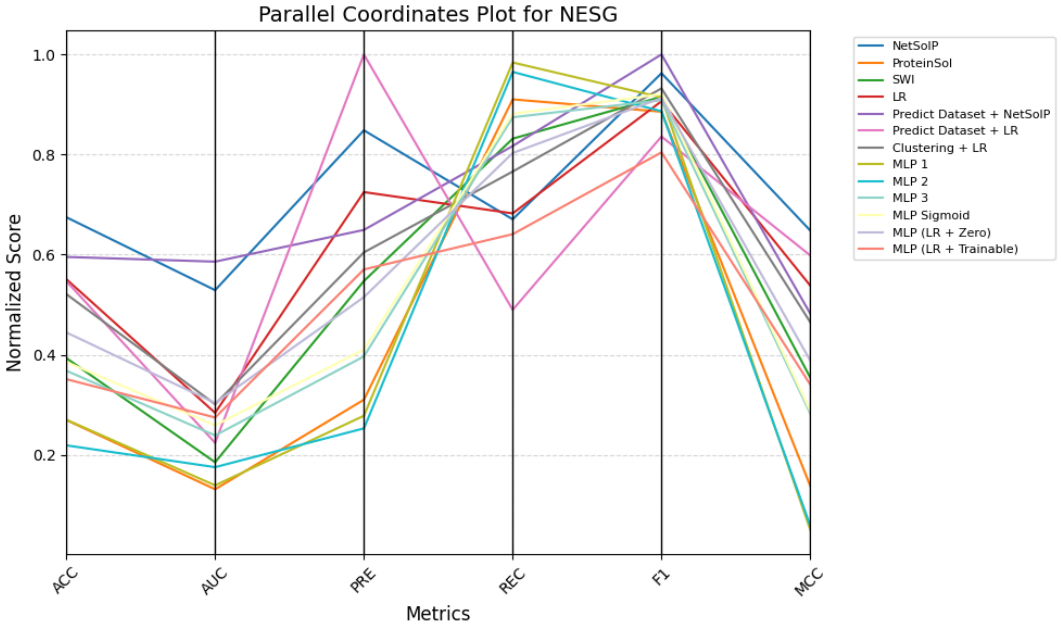


FIGURE 5.29: Shows the normalized metrics for models applied on the NESG dataset.

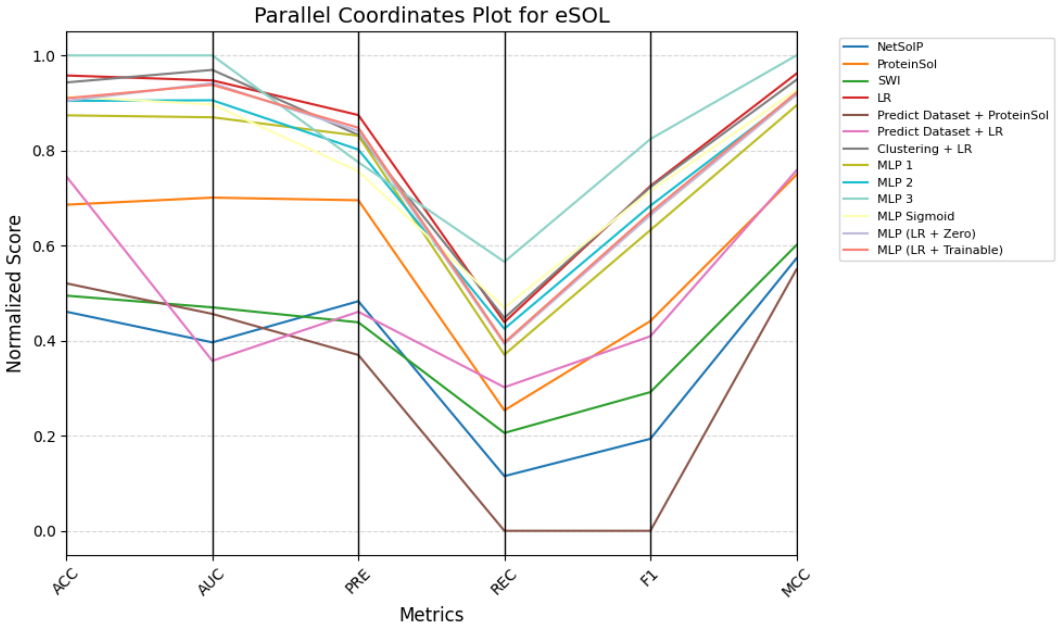


FIGURE 5.30: Shows the normalized metrics for models applied on the eSOL dataset.

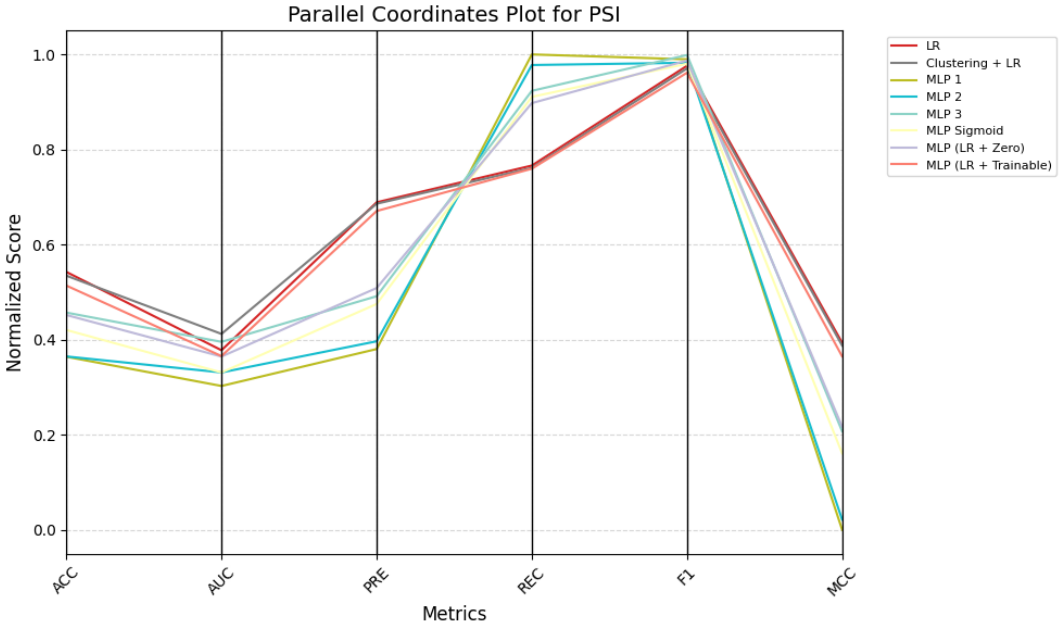


FIGURE 5.31: Shows the normalized metrics for models applied on the PSI: Biology dataset.

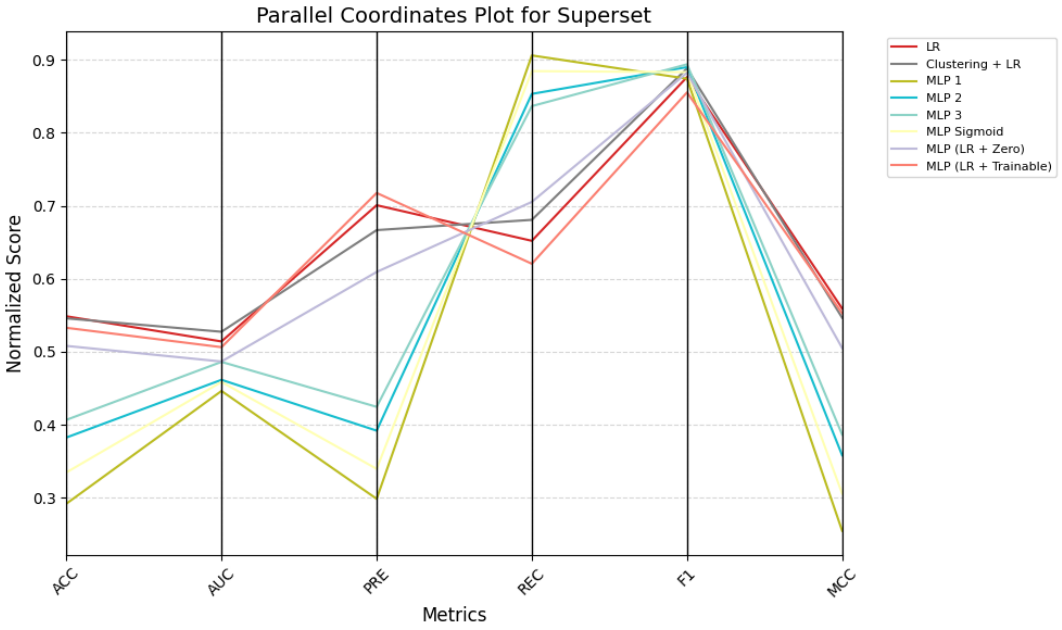


FIGURE 5.32: Shows the normalized metrics for models applied on the Superset dataset.

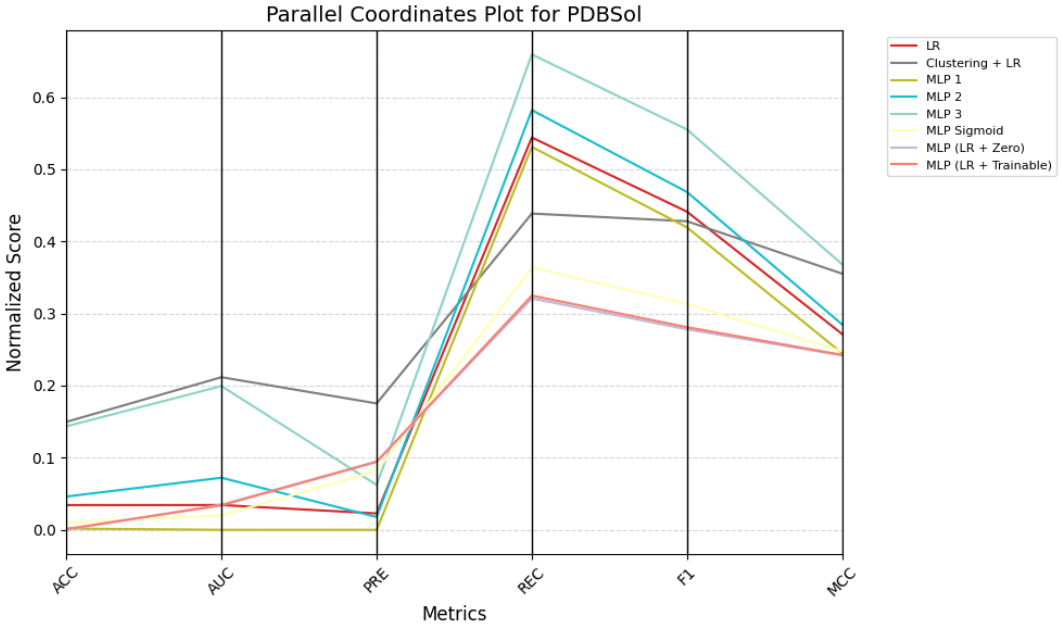


FIGURE 5.33: Shows the normalized metrics for models applied on the PDBSol dataset.

CHAPTER 6

Discussion

6.1 Results Analysis

6.1.1 Dataset Analysis

The classification results from the dataset confusion matrices highlight not only the ability for logistic regression to separate datasets based on sequence-derived features, but also the degree of redundancy and overlap in the data sources themselves. This overlap plays a major role in the observed classification performance. When comparing the accuracy of confusion matrices involving different number of datasets, it is important to note that the classification task becomes more difficult as the number of dataset labels increases.

The clearest separation was achieved when classifying Thera-SAbDab heavy and light chain sequences against those from NESG and eSOL (Figure 5.1), with an overall accuracy of 0.84. This indicates that therapeutic antibody sequences are distinctly different from generic protein sequences. This is expected, as Thera-SAbDab proteins originate from a curated subset of antibody structures derived from the PDB and are heavily optimized for therapeutic applications. As such, their biophysical and sequence properties likely differ significantly from proteins in NESG and eSOL, which are more biologically diverse and not biased toward therapeutic design. Importantly, the confusion matrix shows that the Thera-SAbDab heavy and light chain sequences were classified with almost perfect accuracy, with virtually no misclassifications. The reduced overall accuracy is mainly due to the overlap between NESG and eSOL samples, which

were more frequently confused with one another, suggesting a degree of similarity in their features.

Conversely, NESG and PSI: Biology were difficult to distinguish (Figure 5.2), with an accuracy of just 0.52, close to random chance. This result is expected due to their shared origin. Both datasets were curated from the TargetTrack database, though they differ slightly in how solubility labels were assigned. The strong feature overlap indicates that despite being treated as distinct datasets, they likely represent very similar underlying protein distributions, making them effectively indistinguishable using simple sequence-derived features alone. The comparison between eSOL and PSI: Biology (Figure 5.3) yielded a moderate accuracy of 0.72. Although eSOL is not derived from TargetTrack, it still likely shares general characteristics with PSI: Biology, similarly to the overlap with NESG.

In the multiclass case using NESG, eSOL, PSI: Biology, and PDBSol (Figure 5.4), performance is even lower (accuracy: 0.43), highlighting the overall similarity among datasets that draw from overlapping sources such as TargetTrack and PDB. In particular, PDBSol includes proteins from TargetTrack, directly incorporating data similar to PSI: Biology and NESG. This makes it unsurprising that these datasets are not easily separable and there is a high chance of redundancy in sequence and label distribution due to shared origins.

The accuracy (0.66) when classifying between PSI: Biology and PDBSol (Figure 5.5) could be due to the fact that PDBSol contains a more diverse set of proteins. Unlike PSI: Biology, which comes only from TargetTrack, PDBSol also includes proteins from other sources like UniProtKB and PRSP-2k. This added variety likely makes the PDBSol dataset more distinct, making it easier for the model to tell the two datasets apart in the binary case even though there is some overlap in the datasets.

Overall, these results reflect how the reuse of major resources like TargetTrack and PDB impacts classification outcomes. The more overlap in origin, the more difficult it becomes to distinguish datasets using sequence features. This insight underscores the importance of understanding dataset origin and redundancy when comparing model performance or attempting dataset generalization.

6.1.2 Existing Models

The evaluation of NetSolP, SWI, and ProteinSol on the NESG and eSOL datasets produced results consistent with expectations and prior results found in [70]. NetSolP showed the best overall performance on the NESG dataset, which is not surprising given the similarities between

NESG and PSI: Biology, the latter being a split from a larger dataset used for training NetSolP. Conversely, ProteinSol achieved the highest metrics on the eSOL dataset, reflecting the fact that it was constructed using data derived from eSOL itself. This alignment supports the correct implementation of each model and validates the chosen evaluation pipeline.

Scatterplots comparing model predictions on NESG (Figure 5.7) reveal a moderate degree of noise, with some weak linear trends visible, suggesting partial agreement in solubility estimation. The corresponding heatmap (Figure 5.8) confirms this by showing moderate levels of binary classification agreement. On the other hand, model predictions on the eSOL dataset (Figure 5.10) exhibit weaker visual correlation and appear more scattered, indicating less consistency in how models rank protein solubility and exhibiting more randomness. Still, the prediction agreement heatmap (Figure 5.11) shows some binary agreement.

Application of the models to the Thera-SAbDab dataset yielded the least consistent results. Since this dataset lacks ground truth solubility labels, only relative trends between heavy and light chain predictions could be evaluated. The scatterplots for Thera-SAbDab (Figures 5.12 and 5.14) show that the solubility thresholds derived from the NESG and eSOL datasets fail to meaningfully separate the data, indicating poor generalization of the existing models to antibody sequences. Binary agreement between models (Figures 5.13 and 5.15) is also inconsistent, either artificially high due to most samples clustering in a single quadrant, or low due to differing predictions. This reinforces the conclusion that these models yield unreliable and inconsistent predictions for therapeutic antibody proteins.

These observations are consistent with the earlier dataset analysis, which showed that Thera-SAbDab sequences differ significantly from those in NESG and eSOL. The lack of agreement and poor threshold transferability suggest that the unique sequence and structural characteristics of therapeutic antibodies fall outside the domain these models were trained on.

6.1.3 Created Models

The results from the logistic regression and SVM models with feature selection (Figure 5.16) indicates that accuracy is not strongly affected by the use of feature selection. Neither the feature selection method nor the number of features affect performance very much. As seen in the feature definitions in Section 2.2.1, several features are not independent. This, along with the observed results, suggests a high degree of correlation among the features. That is, some features are redundant, and altogether, they do not express as much information as first thought. This model was trained on PSI: Biology and evaluated on NESG, and performed similarly to models trained and evaluated solely on NESG, a result that differs from initial expectations.

Models were expected to perform better when trained and evaluated on the same dataset. The fact that NESG and PSI contain similar proteins is a probable factor in the observed result. However, further testing is required to draw definitive conclusions about this.

As shown in Figure 5.20, analysis of the logistic regression model coefficients reveals the most influential features in descending order of importance: feature 35 (SWI), 30 (absolute charge), 33 (sequence entropy), 12 (amino acid Leucine, L), 2 (sequence length), 18 (amino acid Serine, S), and 9 (amino acid Histidine, H). The weights show varying degrees of correlation across datasets, as illustrated in Figure 5.19. The NESG, eSOL, PSI: Biology and Superset datasets found mostly similar patterns in how the features relate to solubility, while the PDBSol dataset stands out with notably different coefficient values. This suggests that the feature patterns related to solubility in PDBSol differ significantly from the other datasets, possibly due to its greater diversity.

A comparison was made between two approaches: using logistic regression for dataset prediction and then applying a corresponding solubility model, versus directly applying each model to the dataset it was trained for. Specifically, this involved comparing the performance of NetSolP when applied to proteins predicted to belong to NESG versus when directly applied to NESG, and similarly for ProteinSol on eSOL. The results showed that dataset prediction did not improve performance. NetSolP achieved similar results with and without dataset prediction, while ProteinSol performed worse when using the dataset prediction, as seen in Figures 5.6, 5.9. When using dataset prediction, NetSolP outperformed logistic regression, while logistic regression outperformed ProteinSol, as seen in Figures 5.6, 5.9, and 5.21. This outcome can partly be explained by the fact that ProteinSol is a relatively simple model based on a small subset of the eSOL dataset, which may limit its generalizability. As a result, it is more sensitive to variation and performs poorly when evaluated on proteins predicted to belong to eSOL but that actually originate from NESG. In contrast, NetSolP is more complex and is likely a more generalized model, maintaining performance when applied to proteins from slightly different sources.

The clustering-based logistic regression model showed differing effect of the number of clusters depending on the dataset, as seen in Figure 5.23. For NESG and eSOL, increasing the number of clusters led to a marginal decrease in accuracy and AUC, while PSI: Biology and the Superset were virtually unaffected. In contrast, PDBSol got a slight increase in accuracy and AUC with more clusters. These findings may be explained by differences in dataset characteristics. The negative effect of increasing the number of clusters when training on NESG and eSOL may be due to over-segmentation, smaller sized datasets and more clusters result in fewer data points for the models to generalize well and cause overfitting. The other datasets are larger and are not as affected by this. PDBSol is the largest one and also consists of proteins from four diverse databases, leading to more diverse proteins benefiting from clustering.

The three standard MLPs produced overall similar results and differed somewhat from the other models. However, clear differences between the models were observed, with the complexity of the model reflecting the performance, as seen in Figure 5.25. The most noticeable difference was observed with the PDBSol dataset where MLP 3 with the largest network outperformed all the other models by a noticeable margin. This may be due to the greater diversity of proteins in the PDBSol dataset, which includes proteins from multiple sources, requiring a larger network to interpret and model effectively.

The simpler MLPs developed for comparison with logistic regression performed similarly to, and in some cases slightly better than the deep MLPs, and achieved comparable results to logistic regression, as anticipated. It was also observed that the 2-1 MLP initialized with logistic regression weights for one of the hidden neurons and zero weights for the other, did not significantly improve performance when the zeroed neuron was made trainable, as seen when comparing Figures 5.27 and 5.28. This suggests that the logistic regression model already captured the information available from the features. As a result, training the other neuron did not lead to substantial weight updates or improved modeling since the model was already optimized. Given the simplicity of the network, it is likely that no further patterns could be extracted beyond those already captured by the logistic regression. This result may have come out differently if other, less correlating, features expressing more information had been used.

6.1.4 Comparison of Models

The performance of the models varied across the datasets and no single model consistently outperformed the others. This section discusses the best performing models for each dataset and reflects on trends observed in the results.

NESG. The best performing models on the NESG dataset (Figure 5.29) were NetSolP and the combination of dataset prediction and applying NetSolP. This result is not surprising, as NetSolP is a complex model trained on proteins from the same database. Logistic regression also showed strong performance on NESG.

eSOL. For the eSOL dataset, the best performing model (Figure 5.30) was MLP 3, followed by logistic regression and several other models. Overall, many models performed well on eSOL compared to the other datasets.

PSI: Biology. The best performing models for the PSI: Biology dataset (Figure 5.31) were logistic regression, clustering-based logistic regression, and the MLP (LR weights + trainable neuron). This shows the strength of models based on logistic regression on PSI: Biology .

Superset. On the Superset, the highest performance (Figure 5.32) was achieved by clustering-based logistic regression, logistic regression and MLP (LR weights + trainable neuron). Unsurprisingly, the Superset had very similar result to PSI: Biology, as PSI: Biology makes up a big part of the Superset.

PDBSol. For PDBSol, MLP 3 achieved the best performance, followed by clustering-based logistic regression, MLP 2, and logistic regression (Figure 5.33). However, MLP 3 outperformed the others by a noticeable margin, likely because of its greater model complexity benefited on more diverse proteins.

General Observations. ProteinSol does not perform that well compared to the other models. Most notable is that not even on the eSOL dataset from which ProteinSol derived its weights, does it perform as well as most of the created models. The model performance on the eSOL dataset was considerably higher than the other datasets for most models. This may be a consequence of how eSOL proteins were expressed in an in vitro environment unlike how the other proteins in the other datasets were expressed in an in vivo environment when their solubility was evaluated. The results show that even relatively simple models can perform well on solubility prediction tasks, but when more diverse data is used it can be beneficial to use more complex models that can better interpret and find patterns in the data.

6.2 Further Work

The results and discussion suggest that current solubility prediction models tend to generalize poorly when applied specifically to antibody protein sequences. This limitation highlights a potential direction for future research: the development and evaluation of models trained directly on antibody-related data. One promising approach could be to take advantage of existing solubility models and apply transfer learning techniques, where these models are fine-tuned using antibody-specific protein sequences. This could help the models adapt better to the distinct features of antibodies, which may differ considerably from the proteins used in their original training.

As mentioned earlier, the outcomes from both the existing and newly created models appear to be more suggestive than definitive, largely due to their moderate accuracy and performance

metrics. To improve the reliability of predictions, it could be worthwhile to explore ensemble methods. By combining several models, it may be possible to achieve more robust predictions, especially if the models agree on whether a particular protein is soluble or not. Analyzing how often and under what conditions models make similar predictions could also offer valuable insight into their individual strengths and weaknesses. A model ensemble where multiple models independently predict solubility with high confidence is more likely to yield trustworthy results.

Another recurring challenge throughout this work was related to dataset quality and redundancy. Many of the datasets used originated from overlapping or identical sources, leading to the presence of duplicate or highly similar sequences, which likely influenced both training and evaluation in unintended ways. To address this, future work should consider implementing strong strategies of removing duplicates across datasets, or possibly using feature augmentation techniques to reduce the impact of source-specific biases. Feature augmentation could help by introducing controlled variability into the input features, such as slight perturbations, which can encourage models to learn more generalizable patterns and become less reliant on repetitive or dataset-specific input. Additionally, attention should be given to the labeling process for solubility data. In several cases, the methods by which proteins were labeled as soluble or insoluble were not clearly defined, making it difficult to assess the reliability of the ground truth. Improving labeling transparency and consistency would be an important step toward building more dependable models.

In this study, all developed models predicted solubility based on a set of 36 features derived from the amino acid sequences in the datasets. While these features capture several important biochemical properties, there is room to explore additional or alternative features that may enhance prediction accuracy. For instance, incorporating structural information could provide valuable context, especially since solubility is often influenced by how a protein folds and interacts with its environment. Advances in structure prediction tools, such as AlphaFold2, could make this type of feature extraction more accessible and accurate in future work.

Additionally, most models in this study were trained and tested using splits of the same dataset, which, while useful for internal benchmarking, may limit the model's ability to generalize to new and diverse data. This presents an opportunity for further research focused on cross-dataset validation, where models are trained on one dataset and evaluated on another. Such an approach would provide a more realistic assessment of model generalization and could reveal how well solubility predictors perform across differing datasets. Exploring this direction could lead to the development of more robust and transferable models applicable to a broader range of protein types, including therapeutic antibodies.

CHAPTER 7

Conclusion

This thesis set out to explore the potential and limitations of machine learning approaches in predicting antibody solubility through four main objectives. The findings provide insight into the current capabilities of predictive models, the challenges posed by data quality and redundancy, and the complexity in defining robust solutions in this domain.

First, in evaluating the predictive capabilities of existing machine learning algorithms, the study found that while these models can capture broad trends in antibody solubility, their predictive accuracy remains limited. The highest observed accuracy, 0.732, was achieved by ProteinSol on the eSOL dataset, indicating overall performance that falls short of what would be considered reliable for practical deployment for the existing models. Furthermore, as illustrated in Figures 5.12 and 5.14, which compare the predicted solubility scores from the existing models on the Thera-SAbDab dataset, the predictions appear largely uncorrelated. This weak correlation indicates poor generalization to antibody-specific data and highlights the limited reliability of current models in this application. This suggests that the current algorithms offer indicative rather than conclusive predictions, limiting their standalone applicability in real-world settings.

Second, in addressing the objective to design, implement, and optimize AI-based models for improved predictive performance, the research showed that newly developed models achieved performance comparable to, and in some cases exceeding, that of existing tools. Notably, the logistic regression (LR) model, the clustering-based logistic regression model, and all the multilayer perceptrons (MLPs) outperformed ProteinSol on the eSOL dataset across all evaluation

metrics, as shown in Figure 5.30. Interestingly, simpler algorithms, such as the logistic regression model, performed on par with more complex MLP architectures in some cases. This result suggests that increased model complexity does not always yield better predictive accuracy in this application.

Third, in addressing the objective to analyze model performance and input features to identify patterns or redundancies, the study revealed that dataset redundancy is a significant issue in solubility prediction. Many commonly used datasets originate from overlapping or closely related sources, which can artificially inflate model performance. For example, a logistic regression classifier trained to distinguish between the NESG and PSI:Biology datasets achieved an accuracy of only 0.52, indicating substantial similarity between the two. This is unsurprising, given their shared origin. The overlap likely contributed to the strong performance of the NetSolP model on the NESG dataset, as the model had been trained on an extended version of PSI: Biology.

In terms of input features, several were consistently more predictive across datasets when analyzing the logistic regression weight coefficients, even though some inconsistencies could also be observed. Notably, sequence-derived properties such as the Solubility-Weighted Index (SWI), sequence length, and absolute charge showed strong contributions to model performance, as illustrated in Figure 5.20. Finally, model performance was lowest on larger and more diverse datasets like PDBSol, further emphasizing the importance of dataset composition and diversity in evaluating model robustness.

Finally, in seeking to identify the most accurate and robust models across standardized datasets, no single algorithm emerged as a clear outperformer. Performance differences between models were often marginal and could sometimes be attributed to random variables such as data splitting. Moreover, model rankings varied depending on the dataset used for evaluation as shown in the model comparison in Figures 5.29, 5.30, 5.31, 5.32, and 5.33, reinforcing that solubility prediction is highly context-dependent and that robustness across diverse scenarios remains an unresolved challenge.

In summary, while progress has been made in developing and benchmarking machine learning models for antibody solubility prediction, substantial work remains. The findings underscore the need for more diverse, high-quality datasets and improved understanding of model generalization across biological contexts. Continued research in this area is essential for developing reliable tools that can support antibody engineering in practice.

APPENDIX A

Appendix

A.1 NetSolP Model Tokenization

Token	Numerical Value
$< null\_0 >$	0
$< pad >$	1
$< eos >$	2
$< unk >$	3
L	4
A	5
G	6
V	7
S	8
E	9
R	10
T	11
I	12
D	13
P	14
K	15
Q	16
N	17

F	18
Y	19
M	20
H	21
W	22
C	23
X	24
B	25
U	26
Z	27
O	28
.	29
$-$	30
$< null\_1 >$	31
$< cls >$	32
$< mask >$	33
$< sep >$	34

TABLE A.1: Shows the conversion from token to numerical value used for embedding the input in NetSolP. [74]

A.2 Model Weights & Parameters

Feature Selection	No.	Prediction Model	
		Logistic Regression	**SVM**
χ^2	**10**	C: 1	C: 1; γ: auto
	15	C: 0.1	C: 1; γ: auto
	20	C: 0.1	C: 1; γ: auto
	25	C: 1	C: 1; γ: scale
	30	C: 0.1	C: 1; γ: scale
	35	C: 10	C: 1; γ: auto
RFE	**10**	C: 0.1	C: 1; γ: scale
	15	C: 10	C: 1; γ: scale
	20	C: 10	C: 1; γ: scale
	25	C: 1	C: 1; γ: scale
	30	C: 0.1	C: 1; γ: auto
	35	C: 1	C: 1; γ: auto
Random Forest	**10**	C: 1	C: 1; γ: auto
	15	C: 10	C: 1; γ: scale
	20	C: 0.1	C: 1; γ: auto
	25	C: 10	C: 1; γ: scale
	30	C: 1	C: 1; γ: auto
	35	C: 10	C: 1; γ: scale
Mutual Information	**10**	C: 10	C: 1; γ: auto
	15	C: 10	C: 1; γ: scale
	20	C: 0.1	C: 1; γ: scale
	25	C: 10	C: 1; γ: scale
	30	C: 0.1	C: 1; γ: auto
	35	C: 10	C: 1; γ: auto
All Features	**36**	C: 10	C: 1; γ: scale

FIGURE A.1: Shows the parameters chosen from grid search for the logistic regression and SVM models using feature selection. Results from these models are found in figures 5.16 and A.3.

	Train dataset					
Feature	**NESG**	**eSOL**	**PSI**	**Superset**	**PDBSol**	**Common**
K-R	0	+	+	+		
D-E			0	0		
Sequence length	-	-	-	-	0	
A	+		0	0	0	
C	-			0	+	
D			0	0	0	
E	0		0	0	0	0
F	0		0	0		
G		0	0	0		
H		-	-	-	+	*
I		0	0	0		
K			+	0	+	
L	-		-	-	-	*
M	0	0	0	0	0	0
N	+	0	0	0		
P	-		0	0		
Q	0		0	0		
R	0		0	0	0	
S	-		-	-	-	*
T			0	0	0	
V		+	0	0		
W	0	0	+	0		
Y		-	0	0		
K+R	+	0	0	0		
D+E	0	+	0	+		
K+R-D-E		0	0	0	+	
K+R+D+E	0		0	0	0	0
F+W+Y	0		0	0	0	0
Isoelectric Point		-	0	0	-	
Hydropathy[a]	0	0	0	0	-	0
Absolute Charge[b]	+	+	+	+		*
Fold Propensity	0	0	0	0		0
Disorder	0		0	+	+	
Sequence Entropy		-	-	-	-	*
β-strand Propensity	0			0		
SWI	+	+	+	+		*

FIGURE A.2: Shows the average top 5 positive and negative feature weights as well as feature weights averaged close to zero (in the range $[-10^{-3}, 10^{-3}]$) for each logistic regression model over 5-fold cross-validation. The last column shows the weights that were most consistently positive and negative (marked with *), and zero over all models. (a - Kyte Doolittle scale, b - at pH 7). A visual plot of the weights is found in figure 5.20.

A.3 Feature Selection Results

Feature Selection	Number of Features	Prediction Model	
		Logistic Regression	SVM
χ^2	**10**	0.693	0.690
	15	0.694	0.683
	20	0.687	0.679
	25	0.690	0.681
	30	0.694	0.684
	35	0.691	0.688
Mutual Information	**10**	0.692	0.689
	15	0.691	0.684
	20	0.694	0.689
	25	0.691	0.683
	30	0.692	0.688
	35	0.690	0.689

FIGURE A.3: Shows best accuracy for different combinations of prediction models and feature selection on NESG dataset with 1,323 sequences. Model is trained on PSI Biology dataset with 11,226 sequences. Results using feature selection methods *Recursive Feature Elimination* and *Random Forest* are found in figure 5.16. Exact model parameters from grid search are found in figure A.1.

A.4 Clustering and Logistic Regression Results

Here the results of comparing the different cluster classifiers using the data sets are provided. The baseline is effectively the model using one cluster, taken from the logistic regression model.

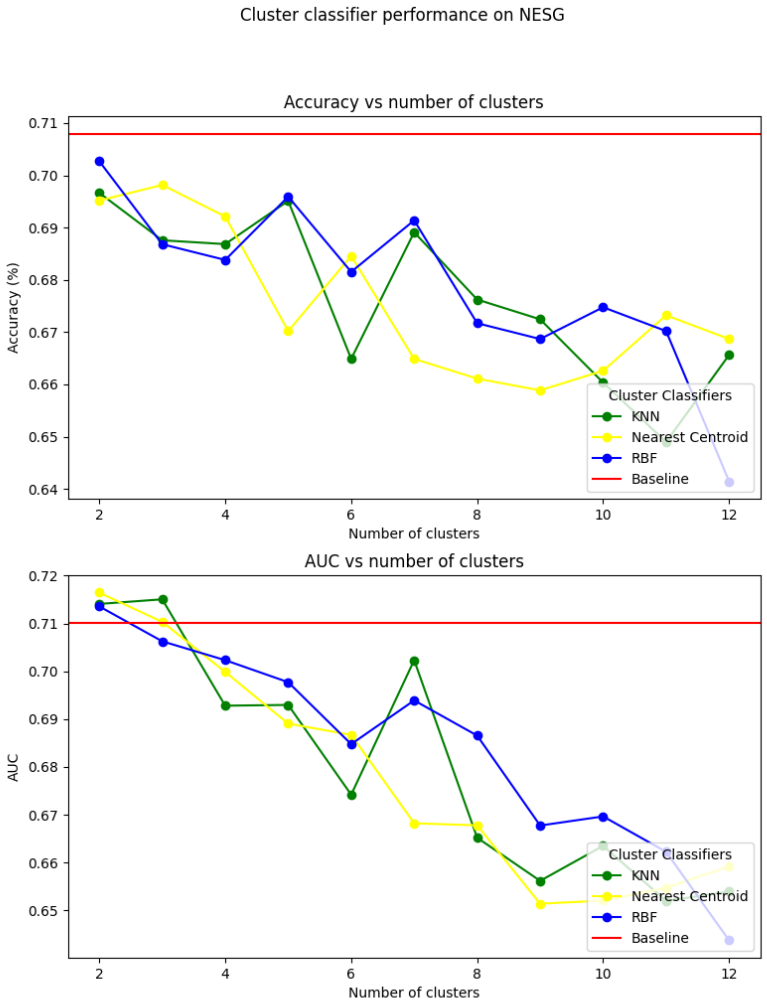


FIGURE A.4: Plot of accuracy and area under ROC curve (AUC) against the number of clusters, from clustering and logistic regression model. KNN, closest centroid and RBF was used as cluster classifiers. The NEGS's training and validation sets were used for clustering and training, while its test set was used for evaluation. The baseline is the value from the logistic regression model on NESG.

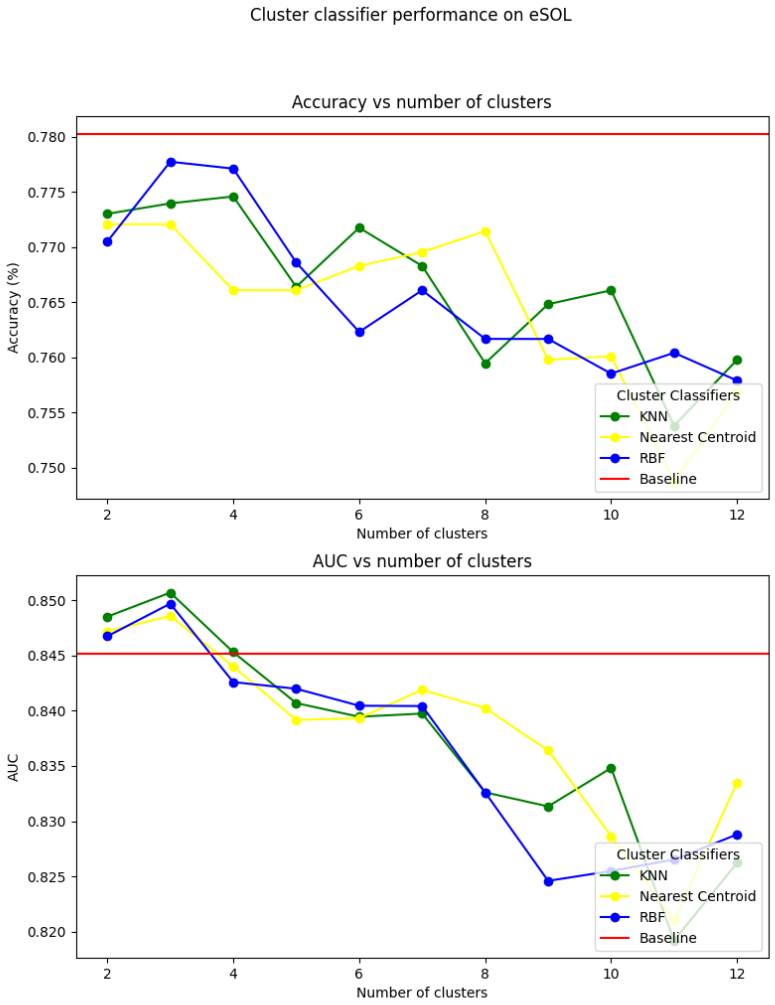


FIGURE A.5: Plot of accuracy and area under ROC curve (AUC) against the number of clusters, from clustering and logistic regression model. KNN, closest centroid and RBF was used as cluster classifiers. The eSOL's training and validation sets were used for clustering and training, while its test set was used for evaluation. The baseline is the value from the logistic regression model on eSOL.

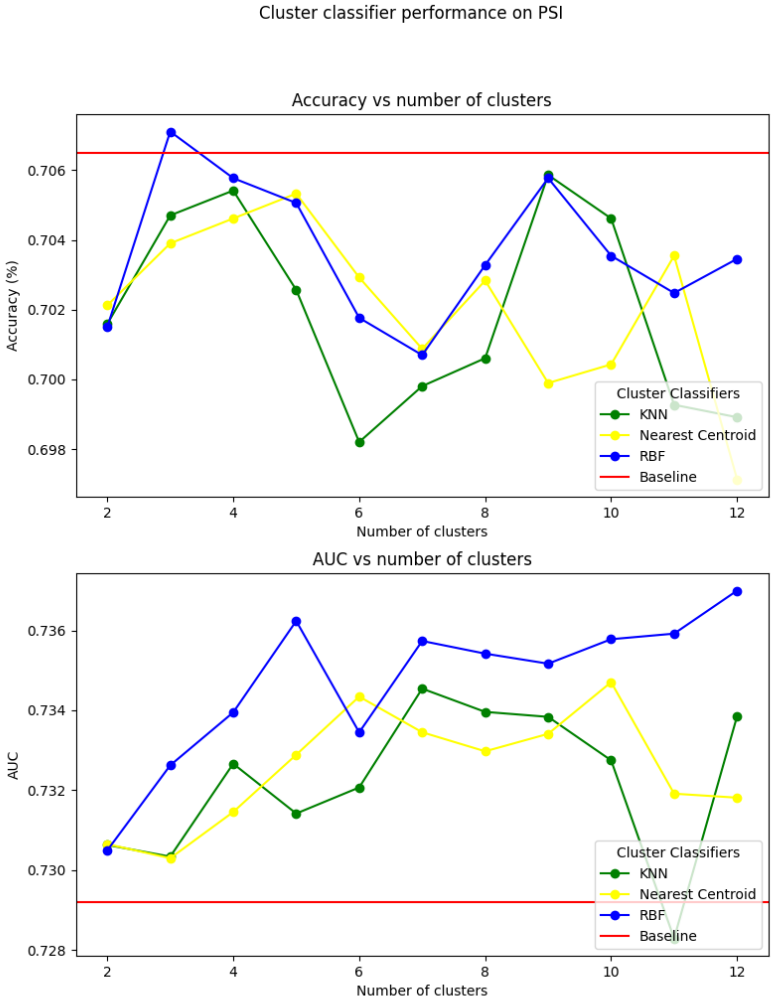


FIGURE A.6: Plot of accuracy and area under ROC curve (AUC) against the number of clusters, from clustering and logistic regression model. KNN, closest centroid and RBF was used as cluster classifiers. The PSI: Biology's training and validation sets were used for clustering and training, while its test set was used for evaluation. The baseline is the value from the logistic regression model on PSI.

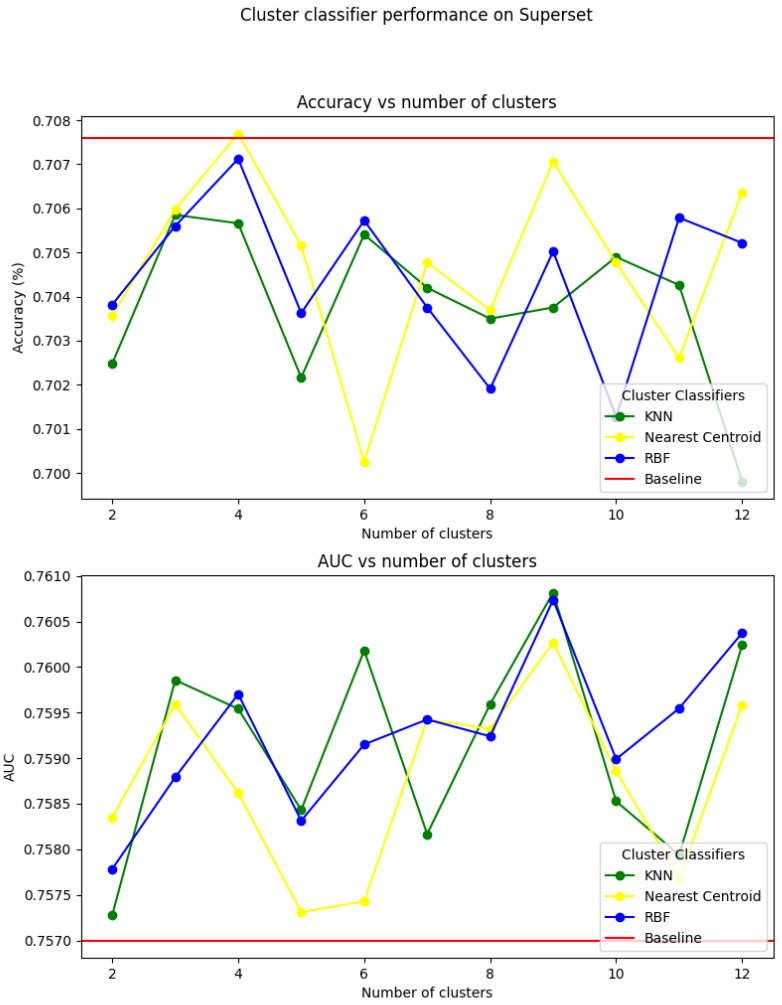


FIGURE A.7: Plot of accuracy and area under ROC curve (AUC) against the number of clusters, from clustering and logistic regression model. KNN, closest centroid and RBF was used as cluster classifiers. The Superset's training and validation sets were used for clustering and training, while its test set was used for evaluation. The baseline is the value from the logistic regression model on the Superset.

Bibliography

[1] Lu R-M. et al. Development of therapeutic antibodies for the treatment of diseases. *Journal of Biomedical Science*, 27(1), 2020. doi: 10.1186/s12929-019-0592-z.

[2] The Antibody Society. Antibody therapeutics product data, n.d.. Retrieved March 26, 2025, from https://www.antibodysociety.org/antibody-therapeutics-product-data/.

[3] The Antibody Society. Antibody therapeutics approved or in regulatory review in the eu or us, n.d.. Retrieved March 26, 2025, from https://www.antibodysociety.org/resources/approved-antibodies/.

[4] Future Market Insights. Antibodies market report, n.d. Retrieved March 26, 2025, from https://www.futuremarketinsights.com/reports/antibodies-market.

[5] Precedence Research. Antibody therapy market size, share & trends — report 2034, 2024. Retrieved from https://www.precedenceresearch.com/antibody-therapy-market.

[6] Zhang W. et al. Developability assessment at early-stage discovery to enable development of antibody-derived therapeutics. *Antibody Therapeutics*, 6(1):13–29, 2022.

[7] Pindrus M. et al. Solubility challenges in high concentration monoclonal antibody formulations: Relationship with amino acid sequence and intermolecular interactions. *Molecular Pharmaceutics*, 12(11):3896–3907, 2015. doi: 10.1021/acs.molpharmaceut.5b00336.

[8] Yang X. et al. Developability studies before initiation of process development: Improving manufacturability of monoclonal antibodies. *MAbs.*, 5(5):787–794, 2013. doi: 10.4161/mabs.25269.

[9] Kenlay H. et al. Large scale paired antibody language models. *PLoS Comput Biol.*, 20(12), 2024. doi: 10.1371/journal.pcbi.1012646.

[10] McCann K. Alphafold 2: The ai system that won google a nobel prize. *AI Magazine*, 2024.

[11] The Editors of Encyclopaedia Britannica. Antibody, 2025. Last updated 12 Feb. 2025. Retrieved from https://www.britannica.com/science/antibody, Accessed 13 February 2025.

[12] The Editors of Encyclopaedia Britannica. Protein, 2025. Last updated 12 Feb. 2025. Retrieved from https://www.britannica.com/science/protein. Accessed 13 February 2025.

[13] National Human Genome Research Institute. Protein structures, 2025. URL https://www.genome.gov/. Image accessed on 2025-03-04.

[14] Lu R. et al. Development of therapeutic antibodies for the treatment of diseases. *Journal of Biomedical Science*, 27(1), 2020. doi: 10.1186/s12929-019-0592-z.

[15] ElBakri A. et al. The state of antibody therapy. *Human Immunology*, 71(12):1243–1250, 2010. doi: 10.1016/j.humimm.2010.09.007.

[16] Otsubo R. and Yasui T. Monoclonal antibody therapeutics for infectious diseases: Beyond normal human immunoglobulin. *Pharmacol Ther*, 240(108233), 2022. doi: 10.1016/j.pharmthera.2022.108233.

[17] Rosace A. et al. Automated optimisation of solubility and conformational stability of antibodies and proteins. *Nature Communications 14:1937*, 14(1), 2023. doi: 10.1038/s41467-023-37668-6.

[18] Xu Y. et al. Structure, heterogeneity and developability assessment of therapeutic antibodie. *MAbs*, 11(2):239–264, 2019. doi: 10.1080/19420862.2018.1553476.

[19] Willis L.F. et al. Biophysical analysis of therapeutic antibodies in the early development pipeline. *Biologics*, 18:413–432, 2024. doi: 10.2147/BTT.S486345.

[20] Halle B. Flexibility and packing in proteins. *Proc. Natl. Acad. Sci. U.S.A*, 99(3):1274–1279, 2002. doi: 10.1073/pnas.032522499.

[21] Bhandari B.K. et al. Solubility-weighted index: Fast and accurate prediction of protein solubility. *Bioinformatics*, 36(18):4691–4698, 2020. doi: 10.1093/bioinformatics/btaa578.

[22] Ghosh K. and Dill K.A. Computing protein stabilities from their chain lengths. *Proceedings of the National Academy of Sciences of the United States of America*, 106(26):10649–10654, 2009. doi: 10.1073/pnas.0903995106.

[23] Kiraga J. et al. The relationships between the isoelectric point and: length of proteins, taxonomy and ecology of organisms. *BMC Genomics*, 8(163), 2007. doi: 10.1186/1471-2164-8-163.

[24] Lopes I. et al. Gene size matters: An analysis of gene length in the human genome. *Frontiers in Genetics*, 12, 2021. doi: 10.3389/fgene.2021.559998.

[25] Kozlowski L.P. Ipc 2.0: prediction of isoelectric point and pka dissociation constants. *Nucleic Acids Research*, 49(W1), 2021. doi: 10.1093/nar/gkab295.

[26] Kyte J. and Doolittle R.F. A simple method for displaying the hydropathic character of a protein. *J. Mol. Biol.*, 157(1):105–132, 1982.

[27] Hebditch M. et al. Protein–sol: A web tool for predicting protein solubility from sequence. *Bioinformatics*, 33(19), 2017. doi: 10.1093/bioinformatics/btx345.

[28] Hebditch M. and Warwicker J. Charge and hydrophobicity are key features in sequence-trained machine learning models for predicting the biophysical properties of clinical-stage antibodies. *PeerJ*, 7, 2019. doi: 10.7717/peerj.8199.

[29] Fujiwara K., Toda H., and Ikeguchi M. Dependence of -helical and -sheet amino acid propensities on the overall protein fold type. *BMC Structural Biology*, 12(1):18, 2012. doi: 10.1186/1472-6807-12-18.

[30] Vladimir N.U. et al. Why are "natively unfolded" proteins unstructured under physiologic conditions? *PROTEINS: Structure, Function, and Genetics*, 41(3):415–427, 2000. doi: 10.1002/1097-0134(20001115)41:3⟨415::aid-prot130⟩3.0.co;2-7.

[31] Linding R. et al. Globplot: exploring protein sequences for globularity and disorder. *Nucleic Acids Research*, 31(13):3701–3708, 2003. doi: 10.1093/nar/gkg519.

[32] Stahura F.L. et al. Differential shannon entropy analysis identifies molecular property descriptors that predict aqueous solubility of synthetic compounds with high accuracy in binary qsar calculations. *Journal of Chemical Information and Computer Sciences*, 42(3), 2002. doi: 10.1021/ci010243q.

[33] Costantini S., Colonna G., and Facchiano A.M. Amino acid propensities for secondary structures are influenced by the protein structural class. *Biochemical and Biophysical Research Communications*, 342(2):441–451, 2006. doi: https://doi.org/10.1016/j.bbrc.2006.01.159.

[34] National Human Genome Research Institute. Antibody. Image retrieved April 4, 2025, from https://www.genome.gov/genetics-glossary/Antibody.

[35] Asmani A.Z.A. et al. Immunogenicity of monoclonal antibody: Causes, consequences, and control strategies. *Pathol Res Pract*, 263(155627), 2024. doi: 10.1016/j.prp.2024.155627.

[36] Murphy K.P. *Machine Learning: a Probabilistic Perspective*, chapter 1 Introduction, pages 1–25. MIT Press, Cambridge, Massachusetts, 2012.

[37] Mohri M., Rostamizadeh A., and Talwalkar A. *Foundations of Machine Learning: Second Edition*, chapter 1 Introduction, pages 1–8. MIT Press, Cambridge, Massachusetts, 2018.

[38] Goodfellow I., Bengio Y., and Courville A. *Deep Learning*, chapter 5 Machine Learning Basics, pages 96–161. MIT Press, Cambridge, Massachusetts, 2016.

[39] GeeksforGeeks. K-nearest neighbor(knn) algorithm. https://www.geeksforgeeks.org/k-nearest-neighbours/, . Retrieved May 15, 2025.

[40] GeeksforGeeks. Ml - nearest centroid classifier. https://www.geeksforgeeks.org/ml-nearest-centroid-classifier/, . Retrieved May 15, 2025.

[41] Pampel F.C. *Logistic Regression: A Primer*. SAGE Publications, Inc., Thousand Oaks, 2021. https://methods.sagepub.com/book/mono/logistic-regression-2e/toc.

[42] LaValley M.P. Logistic regression. *Circulation*, 117(18):2395–2399, 2008. doi: 10.1161/CIRCULATIONAHA.106.682658.

[43] Murphy K.P. *Machine Learning: a Probabilistic Perspective*, chapter 13.3 l1 regularization: basics, pages 429–440. MIT Press, Cambridge, Massachusetts, 2012.

[44] Mohri M., Rostamizadeh A., and Talwalkar A. *Foundations of Machine Learning: Second Edition*, chapter 5 Support Vector Machines, pages 79–104. MIT Press, Cambridge, Massachusetts, 2018.

[45] Mohri M., Rostamizadeh A., and Talwalkar A. *Foundations of Machine Learning: Second Edition*, chapter 6 Kernel Methods, pages 105–144. MIT Press, Cambridge, Massachusetts, 2018.

[46] Ikotun A.M. et al. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210, 2023. doi: https://doi.org/10.1016/j.ins.2022.11.139.

[47] Khurana D. et al. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82:3713–3744, 2023. doi: 10.1007/s11042-022-13428-4.

[48] Manning C. et al. *An Introduction to Information Retrieval*, chapter 2, pages 19–47. Cambridge University Press, Cambridge, England, 2009.

[49] Viswanathan K. et al. The geometry of tokens in internal representations of large language models, 2025. URL https://arxiv.org/abs/2501.10573.

[50] Murphy K.P. *Machine Learning: a Probabilistic Perspective*, chapter 3.5.3 The log-sum-exp trick, pages 86–87. MIT Press, Cambridge, Massachusetts, 2012.

[51] Manning C. et al. *An Introduction to Information Retrieval*, chapter 13.5 Feature Selection, pages 271–279. Cambridge University Press Cambridge, Cambridge, England, 2009.

[52] scikit-learn. Rfe — recursive feature elimination, . Retrieved April 14, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html.

[53] scikit-learn. Randomforestclassifier, . Retrieved April 14, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.

[54] Murphy K.P. *Machine Learning: a Probabilistic Perspective*, chapter 16.2.2.2 Classification cost, pages 547–548. MIT Press, Cambridge, Massachusetts, 2012.

[55] Mohri M., Rostamizadeh A., and Talwalkar A. *Foundations of Machine Learning: Second Edition*, chapter 4 Model Selection, pages 61–78. MIT Press, Cambridge, Massachusetts, 2018.

[56] Bischl B. et al. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(4), 2023. doi: 10.1002/widm.1484.

[57] scikit-learn. Tuning the hyper-parameters of an estimator — grid search, . Retrieved April 15, 2025, from https://scikit-learn.org/stable/modules/grid_search.html.

[58] Roberts D.A., Yaida S., and Hanin B. *The Principles of Deep Learning Theory*, chapter 2 Neural Networks, pages 37–52. Cambridge University Press, Cambridge, England, 2022.

[59] Aggarwal C.C. *Neural Networks and Deep Learning: A Textbook*, chapter 1 An Introduction to Neural Networks, pages 1–51. Springer Nature Switzerland AG, Cham, Switzerland, 2018.

[60] Eunice Kennedy Shriver National Institute of Child Health and Human Development. What are the parts of the nervous system?, 2025. URL https://www.ninds.nih.gov/what-are-parts-nervous-system. Image accessed on 2025-03-04.

[61] Aggarwal C.C. *Neural Networks and Deep Learning: A Textbook*, chapter 4 Deep Learning: Principles and Training Algorithms, pages 119–163. Springer Nature Switzerland AG, Cham, Switzerland, 2018.

[62] Aggarwal C.C. *Neural Networks and Deep Learning: A Textbook*, chapter 2 The Backpropagation Algorithm, pages 29–71. Springer Nature Switzerland AG, Cham, Switzerland, 2018.

[63] Aggarwal C.C. *Neural Networks and Deep Learning: A Textbook*, chapter 5 Teaching Deep Learners to Generalize, pages 165–214. Springer Nature Switzerland AG, Cham, Switzerland, 2018.

[64] You K. et al. How does learning rate decay help modern neural networks? *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. arXiv:1908.01878.

[65] Pan S.J. and Yang Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[66] Tan C. et al. A survey on deep transfer learning. In Kůrková V. et al., editors, *Artificial Neural Networks and Machine Learning – ICANN 2018*, volume 11141 of *Lecture Notes in Computer Science*. Springer, Cham, 2018. doi: 10.1007/978-3-030-01424-7_27.

[67] Lindholm A., Wahlström N., Lindsten F., and Schön T.B. *Machine Learning: A First Course for Engineers and Scientists*, chapter 4.5 Additional Tools for Evaluating Binary Classifiers, pages 86–90. Cambridge University Press, 2022.

[68] Zheng A. *Evaluating Machine Learning Models: A Beginner's Guide to Key Concepts and Pitfalls*, chapter 2 Evaluation Metrics. O'Reilly Media, Inc., Sebastopol, CA, USA, 2015.

[69] Chicco D. and Jurman G. The matthews correlation coefficient (mcc) should replace the roc auc as the standard metric for assessing binary classification. *BioData Mining*, 16(4), 2023. doi: 10.1186/s13040-023-00307-5.

[70] Thumuluri V. et al. Netsolp: Predicting protein solubility in escherichia coli using language models. *Bioinformatics*, 38(4):941–946, 2022.

[71] Vaswani A. et al. Attention is all you need. In Guyon I. et al., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[72] Alammar J. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 2018. Accessed 2025-02-13.

[73] Rives A. et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc Natl Acad Sci U S A*, 118(15), 2021. doi: 10.1073/pnas.2016239118.

[74] DTU HealthTech. Netsolp: Code for predicting protein solubility. https://services.healthtech.dtu.dk/services/NetSolP-1.0/. Accessed: 2025-01-20.

[75] Lagarias J. et al. Convergence properties of the nelder–mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998. doi: 10.1137/S1052623495288627.

[76] K. Gonçalves-e Silva et al. Less is more: Simplified nelder-mead method for large unconstrained optimization. *Yugoslav Journal of Operations Research*, 28:14–14, 2018. doi: 10.2298/YJOR180120014G.

[77] Niwa T. et al. Bimodal protein solubility distribution revealed by an aggregation analysis of the entire ensemble of *Escherichia coli* proteins. *Proc Natl Acad Sci U S A*, 106(11): 4201–4206, 2009. doi: 10.1073/pnas.0811922106.

[78] DTU Health Tech. Netsolp-1.0. data. Accessed from https://services.healthtech.dtu.dk/services/NetSolP-1.0/.

[79] Berman H.M. et al. Protein structure initiative—targettrack 2000–2017—all data files, 2017. URL https://zenodo.org/records/821654. Zenodo.

[80] Gardner BinfLab. Sodope_paper_2020. https://github.com/Gardner-BinfLab/SoDoPE_paper_2020, 2020.

[81] SAbDab. Thera-sabdab. Accessed from https://opig.stats.ox.ac.uk/webapps/sabdab-sabpred/therasabdab/search/.

[82] Protein Data Bank. Protein data bank. Accessed from https://www.rcsb.org/.

[83] Yang T. et al. Protsolm: Protein solubility prediction with multi-modal features. *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 223–232, 2024.

[84] Fan R. et al. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9:1871–1874, 2008.

[85] scikit-learn. Kmeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, . Retrieved April 17, 2025.

[86] scikit-learn. Logisticregression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, . Retrieved April 16, 2025.

[87] GPy. Kern.k — gpy rbf kernel function. https://gpy.readthedocs.io/en/devel/GPy.kern.src.html#GPy.kern.src.kern.Kern.K. Retrieved April 17, 2025.

Master's Theses in Mathematical Sciences 2025:E33
ISSN 1404-6342

LUTFMA-3582-2025

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/