FEDERAL UNIVERSITY OF CEARÁ

CENTER OF SCIENCE

DEPARTMENT OF PHYSICS

PAULA JIMENA FONSECA CALA

PREDICTION OF THE SURFACE TENSION OF HYDROCARBONS WITH REGRESSORS BASED ON MACHINE LEARNING TECHNIQUES.

FORTALEZA

2023

PAULA JIMENA FONSECA CALA

PREDICTION OF THE SURFACE TENSION OF HYDROCARBONS WITH REGRESSORS BASED ON MACHINE LEARNING TECHNIQUES.

Dissertation submitted to the Post-Graduation Program in Physics of the Center of Science of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Physics.
Advisor: Prof. Dr. Amuri Jardim De Paula

Co-advisor: Prof. Dr. James Moraes De Almeida

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F746p Fonseca Cala, Paula Jimena.
Prediction Of The Surface Tension Of Hydrocarbons With Regressors Based On Machine Learning Techniques. / Paula Jimena Fonseca Cala. – 2023.
65 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Física, Fortaleza, 2023.
Orientação: Prof. Dr. Amauri Jardim de Paula.
Coorientação: Prof. Dr. James Moraes de Almeida.

1. Tensão superficial. 2. Hidrocarbonetos. 3. Aprendizado de máquina. 4. Algoritmos de árvore de decisão. I. Título.

CDD 530

PAULA JIMENA FONSECA CALA

PREDICTION OF THE SURFACE TENSION OF HYDROCARBONS WITH REGRESSORS BASED ON MACHINE LEARNING TECHNIQUES.

Dissertation submitted to the Post-Graduation Program in Physics of the Center of Science of the Federal University of Ceará, as a partial requirement for obtaining the title of Master in Physics.

Approved on: 06/11/2023

EXAMINATION BOARD

Prof. Dr. Amuri Jardim De Paula (Advisor)
Federal University of Ceará (UFC)

Prof. Dr. James Moraes De Almeida (Co-advisor)
Brazilian Center for Research in Energy and Materials (Ilum-CNPEM)

Prof. Dr. Alexandre Paschoal
Federal University of Ceará (UFC)

Prof. Dr. Daniel Cassar
Brazilian Center for Research in Energy and Materials (Ilum-CNPEM)

A mi familia.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Ph.D. Professors Amauri and James for their invaluable contributions of time, patience, and unwavering dedication to the development of this work. Without their support, this endeavor would not have been possible, and I am truly thankful for this incredible opportunity.

I am also immensely grateful to my parents and brothers for their support and unconditional love throughout this journey; they have been the driving force in my life.

To my dear friends and colleagues, especially Laura Pabón, I want to extend my appreciation for the wonderful moments we shared along this path.

I wish to thank Juan Camilo for his love and trust in me.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

ABSTRACT

Surface tension is a critical parameter in the petrochemical industry. It plays a key role in designing and optimizing various processes related to the production, transportation, and refining of hydrocarbons. The importance of surface tension lies in its ability to influence the behavior of liquids during these processes, affecting factors such as flow rate, separation efficiency, and the stability of emulsions. In recent years, machine learning (ML) techniques have shown promise in predicting the physical and chemical properties of hydrocarbons. These techniques offer a data-driven approach to understanding complex systems, and they have the potential to significantly improve the efficiency and accuracy of predictions related to hydrocarbon properties. In this study, we compared various machine learning algorithms, including K-Nearest Neighbors (KNN), Random Forest (RF), and XGBoost (XGB), to determine their effectiveness in predicting the surface tension of hydrocarbons. These algorithms were chosen due to their popularity and proven effectiveness in a variety of applications. The results of our study indicate that XGBoost exhibited the best performance in predicting the surface tension of hydrocarbons, with a mean squared error (MSE) of 4.65 and an R^2 score of 0.88. The R^2 score, also known as the coefficient of determination, is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. An R^2 score of 0.88 indicates a high level of accuracy in the predictions made by the XGBoost model. This study provides promising evidence that machine learning techniques can be effectively applied to predict the surface tension of hydrocarbons. The successful application of these techniques could lead to significant improvements in the efficiency and accuracy of processes in the petrochemical industry.

Keywords: surface tension; hydrocarbons; machine learning; decision tree algorithms.

RESUMO

A tensão superficial é um parâmetro crítico na indústria petroquímica. Desempenha um papel fundamental na concepção e otimização de diversos processos relacionados à produção, transporte e refino de hidrocarbonetos. A importância da tensão superficial reside na sua capacidade de influenciar o comportamento dos líquidos durante esses processos, afetando fatores como vazão, eficiência de separação e estabilidade das emulsões. Nos últimos anos, as técnicas de aprendizado de máquina (ML) têm se mostrado promissoras na previsão das propriedades físicas e químicas dos hidrocarbonetos. Estas técnicas oferecem uma abordagem baseada em dados para a compreensão de sistemas complexos e têm o potencial de melhorar significativamente a eficiência e a precisão das previsões relacionadas com as propriedades dos hidrocarbonetos. Neste estudo, comparamos vários algoritmos de aprendizado de máquina, incluindo K-Nearest Neighbours (KNN), Random Forest (RF) e XGBoost (XGB), para determinar sua eficácia na previsão da tensão superficial de hidrocarbonetos. Esses algoritmos foram escolhidos devido à sua popularidade e eficácia comprovada em diversas aplicações. Os resultados do nosso estudo indicam que o XGBoost exibiu o melhor desempenho na previsão da tensão superficial de hidrocarbonetos, com um erro quadrático médio (MSE) de 4,65 e uma pontuação R^2 de 0,88. A pontuação R^2, também conhecida como coeficiente de determinação, é uma medida estatística que representa a proporção da variância de uma variável dependente que é explicada por uma variável ou variáveis independentes em um modelo de regressão. Uma pontuação R^2 de 0,88 indica um alto nível de precisão nas previsões feitas pelo modelo XGBoost. Este estudo fornece evidências promissoras de que técnicas de aprendizado de máquina podem ser aplicadas de forma eficaz para prever a tensão superficial de hidrocarbonetos. A aplicação bem sucedida destas técnicas poderá levar a melhorias significativas na eficiência e precisão dos processos na indústria petroquímica.

Palavras-chave: tensão superficial; hidrocarbonetos; aprendizado de máquina; algoritmos de árvore de decisão.

LIST OF FIGURES

Figure 21 – Importance of the characteristics is highlighted for each of the available data sets. It should be noted that the colors of the bars represent different data sets mentioned in the document. This analysis specifically applies to algorithms based on decision trees, such as RF and XGB. It is evident that while they share some of the most important characteristics, they are not exactly the same. 61

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Surface tension is an elemental factor in fluid mechanics, significantly affecting numerous industrial operations. This property of the fluids is essential in various industries including but not limited to petrochemicals (SHENG, 2013; MASSARWEH; ABUSHAIKHA, 2020; TAVAKKOLI *et al.*, 2022), microfluidics (BARET, 2012; LEE *et al.*, 2011), cleaning (COX, 1986; MOHARRAM *et al.*, 2013; CHELAZZI *et al.*, 2020; PIRONTI *et al.*, 2020), medicine (MALDONADO-VALDERRAMA *et al.*, 2011; PERCIVAL *et al.*, 2017; CALLION *et al.*, 1996), and agriculture (PARIA, 2008; HERNÁNDEZ-SORIANO *et al.*, 2011; HERNÁNDEZ-SORIANO *et al.*, 2010). Despite its widespread applications and importance, forecasting the surface tension of organic compounds is a difficult task. This is due to the intricate chemical structure and varied properties of organic compounds (WU *et al.*, 2023).

Surface tension is an important characteristic of organic compounds that profoundly influences their behavior in various industrial applications. Determining this property accurately is essential to numerous fields such as oil extraction, bubble formation, adhesion of liquids to solid surfaces, and the formulation of chemical products (KATZ; SALTMAN, 1939). Traditional techniques for measuring surface tension have been in use since the 1930s, with methods like the capillary rise method, thus providing valuable insights into the nature of surface tension, including the finding that surface tension of gas-saturated oil decreases with increasing saturation pressure (SWARTZ, 1931).

Extending the research, studies involving binary mixtures like krypton, ethane, and ethylene have been conducted using mean field theory (MFT). These studies were successful in drawing a substantial correlation between theoretical and experimental values (ALMEIDA; GAMA, 1989). Innovative measurement techniques have also been proposed by scholars like Abbas and Nordholm, who put forth a technique based on the generalized van der Waals theory (GvdW). For simple polar fluids, this method has been successful in achieving experimental values that closely match theoretical predictions, thus showing a high correlation (SIMPLE. . . , 1995).

As the field continued to grow and evolve, newer, more sophisticated methods have been introduced to the study of surface tension. Fu et al. (FU *et al.*, 2001) proposed the development of a theoretical model built on the foundation of density functional theory (DFT) and the Barker-Henderson perturbation theory (BAKER; HENDERSON, 1967). Utilizing this model on 18 non-pure polar fluids demonstrated that the equation of state proposed by the

model could predict surface tension with a deviation of 3.3%, indicating a fairly high level of accuracy. Such research indicates the increasing complexity and sophistication of methods to predict surface tension.

In recent years, machine learning techniques have seen a surge in popularity, with increasing evidence suggesting their potential in accurately predicting these physical and chemical properties (ZHOU *et al.*, 2020). Machine learning methodologies, utilizing advanced mathematical models, can learn from experimental data, train on it, and then use that training to predict outcomes for new instances. Their proven effectiveness in predicting an array of properties for hydrocarbons, including boiling points, densities, and vapor pressures, showcases their potential (DOBBELAERE *et al.*, 2022). In light of this, this research focuses on promoting the use of machine learning techniques to predict the surface tension of organic compounds.

Recently, the fusion of traditional scientific methods with advanced computational techniques has brought machine learning into the study of surface tension. A proposal for a hybrid model that combines machine learning techniques such as linear regression and neural networks was conducted, aiming to develop a predictive model for determining the surface tension of hydrocarbon surfactants in aqueous solutions. In this regard, the work of Seddon et al. (SEDDON *et al.*, 2022) is significant. They demonstrated that the three parameters of interest from the Szyszkowski equation could be determined using a Python code.

In another innovative approach, Soori et al. (ROKONI; SUN, 2021) proposed the usage of Convolutional Neural Networks (CNN) to estimate surface tension from images of hanging drops with unknown concentrations of ethanol and water. The trained machine learning models achieved high accuracies, ranging between 97.8% and 99.5%. This technique allows for easy, fast, and relatively accurate determination of surface tension, bypassing the need for tedious preprocessing of image databases. It represents a significant advancement in the application of machine learning to the study of surface tension.

This research has successfully delivered several notable accomplishments. Firstly, we have compiled extensive databases containing detailed quantitative information pertaining to thousands of organic compounds (a). Secondly, we've developed an intelligent algorithm that is designed to choose the dataset that could serve as the optimal training set for machine learning algorithms (b). Additionally, we have put together a top-performing regression algorithm that makes the most out of the selected dataset (c). Finally, we conducted an analysis of the factors that hold the greatest significance when it comes to predicting surface tension in hydrocarbons.

2 MACHINE LEARNING TECHNIQUES

Machine learning is a subsection of artificial intelligence that involves the creation of computational models that can learn from data and make predictions or decisions without explicit programming. Machine learning algorithms are typically categorized into three paradigms: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning involves teaching algorithms to make predictions based on labeled data, where both the input and desired output are provided. This learning paradigm is widely used in applications like email filtering, where algorithms are trained to classify emails as spam or not-spam, and regression tasks, such as predicting housing prices based on features like location, size, and age of the house. In supervised learning, the objective is to learn a function denoted as $y = f(x)$ from a given dataset $\{x_i, y_i\}_{i=1}^{n}$. Here, x_i represents the i-th data point, and y_i represents its corresponding label.

Unsupervised learning, on the other hand, deals with algorithms that learn from unlabeled data by identifying patterns or structures within it. These algorithms are often used in applications like customer segmentation in marketing, where the algorithm clusters customers based on their purchasing behavior, and anomaly detection, for instance, in credit card fraud detection, where the algorithm identifies transactions that deviate significantly from the norm. In unsupervised learning, the primary objective is to discover and understand underlying patterns or similarities within a dataset $\{x_i\}_{i=1}^{n}$.

Reinforcement learning is a paradigm where algorithms learn by interacting with their environment, receiving feedback in the form of rewards or penalties. This paradigm is typically applied in areas like game playing, where the algorithm learns to make moves that maximize the game score, and robotics, where a robot learns to perform tasks by maximizing a reward function.

In the development of this research, our focus will be on the supervised learning paradigm. Specifically, we will explore how to make predictions from a dataset.

2.1 Supervised Learning

Within the realm of supervised learning, there exist two principal applications of algorithms: regression and classification. These applications serve distinct functions and are chosen based on the nature of the problem at hand.

Regression is a type of supervised learning task that is utilized when the output variable is continuous or numerical in nature. It focuses on predicting a continuous outcome variable (or a dependent variable) from one or more predictor variables (or independent variables). For instance, predicting the price of a house based on factors such as location, number of rooms, and age is a typical example of a regression problem. Similarly, forecasting stock prices, estimating life expectancy, and predicting sales amounts are other common applications of regression in various fields.

On the other hand, classification is a supervised learning task used when the output variable is categorical. It involves predicting the class or category of a given input. For instance, determining whether an email is spam or not based on its content is a classification problem. Similarly, diagnosing diseases based on patient symptoms, recognizing handwriting, and identifying fraudulent transactions are typical examples of classification problems.

In both regression and classification tasks, the aim is to construct a model that learns from labeled training data in order to make accurate predictions when faced with new, unseen data. Both these applications form the backbone of supervised learning and are vital tools in the machine learning toolkit.

Given the existence of a set of data x_i associated with a label y_i, it is a challenge to analytically find a model S that represents the relationship between labels and data. Therefore, what is done is to estimate this model $\hat{S}$, and the construction of this model is known as learning. The underlying question to this fact is, how do we know how good $\hat{S}$ is? The answer is implicit in the data itself. We start with the hypothesis that the data is modeled in some way and that this model relates it to its labels. Thus, if I estimate new labels $\hat{y}_i$ with the new model $\hat{S}$, a comparison between the models becomes possible. Now, to measure this performance, an error criterion must be applied, which will depend on the task that this model is facing (classification or regression).

In a probabilistic framework, it is known that $(\mathbf{x}, y) \sim D$, which implies that for the classification task, $\mathbf{x} \sim D$ and $y = c(x)$ for some unknown deterministic classifier c. Furthermore, $\mathbf{x} \sim D$ and $P[y = 1|\mathbf{x}] = \alpha(x)$, where $\alpha(x)$ is a mapping fuction from the feature space to the interval $[0,1]$, representing probabilities. In probabilistic terms, this can be expressed as: $\alpha : \mathscr{X} \to [0,1]$, where $\mathscr{X}$ is the feature space, and for each $\mathbf{x} \in \mathscr{X}$, $\alpha(\mathbf{x})$ gives the conditional probability that the corresponding label y is equal to 1, given the feature vector $\mathbf{x}$. If the problem is regression, the assumption that the data follows some distribution still holds, and it can be stated that $\mathbf{x} \sim D$ and $y = f(x) + \eta$, where $\eta \sim D_\eta$, that means that in terms of the regression

framework, the regression consists of a deterministic function added to a noise term. The noise term is introduced as η, and of course, this error term has some distribution, denoted as $\eta \sim D_\eta$. Now, suppose that from the original dataset $\mathbf{x}$, it is possible to obtain two subsets that will be referred to as the training set and the test set, denoted as $\{x_i, y_i\}_{i=0}^{q}$ and $\{x_i, y_i\}_{i=0}^{n}$, respectively. The following conditions must hold:

- $(\mathbf{x}, y) \sim D$: The joint distribution of the data and labels is denoted by D.
- $\{x_i, y_i\}_{i=0}^{n} \sim D$: The training set is independently sampled from D.
- $\{x_i, y_i\}_{i=0}^{q} \sim D$: The test set is independently sampled from D, and it is also independent of the training set $\{x_i, y_i\}_{i=0}^{n}$.

The above allows us to determine how the error will be measured. In the case of regression, the error can be expressed as the expected squared difference between the predictions of the model $\hat{S}(x)$ and the true labels y:

$$\mathbf{E}_D \left[\hat{S}(x) - y\right]^2. \tag{2.1}$$

This means that the error is the average squared difference between the predicted labels and the true labels of the original data. On the other hand, in classification, the error can be defined as:

$$\mathbf{P}_D \left[\hat{S}(x) \neq y\right]. \tag{2.2}$$

This quantifies the error of misclassifying a data point, representing the probability that a data point x_i does not belong to its true class.

So far, we have discussed concepts related to the data and the need to quantify the error of a model, without going into much detail about the type of model $\hat{S}$, which will be explored in subsequent chapters. In later chapters, we will learn that there is a way to validate the model before testing it on unseen data. The process of building a supervised learning model involves several intermediate steps that have not been mentioned yet, and their explanations could be extensive, as each step has its own rationale. However, we will briefly list these steps, and if necessary, provide explanations where needed. The steps involved in generating a supervised machine learning model correctly are as follows:

- Splitting the data into three subsets (Training/Validation/Test).
- Selecting the ML algorithm (Linear Regression, K-Nearest Neighbors, Random Forest, etc.).
- Data preprocessing.

- Training:
  - Choosing an error function.
  - Training models of different "sizes" or complexities:
    - Optimizing the error function on the training data.
    - Evaluating the model on the validation data.
  - Selecting the model with the lowest error on the validation data.
- Evaluating the model on the test data.

These steps outline the general process for building a supervised machine learning model, and each step plays a crucial role in developing an effective and reliable model.

In the following sections you will find the following content:

- **Linear Regression**: In this section, we will explore the most basic form of $\hat{S}(x)$, which is linear regression. Linear regression assumes a linear relationship between the input features and the target variable. The mathematical formulation and interpretation of the coefficients will be discussed.
- **Nearest Neighbors Algorithms**: In this section, we will delve into algorithms based on nearest neighbors, such as k-nearest neighbors (KNN). We will discuss the intuition behind these algorithms and their implementation, including considerations for parameter tuning and distance metrics.
- **Boosting Techniques**: This section will introduce boosting techniques, these techniques combine multiple weak models to create a strong predictive model. We will explain the underlying principles and the ensemble learning process.
- **Tree-Based Models**: In this section, we introduce decision tree-based models, beginning with a broad overview and progressing to essential theoretical considerations. These foundational concepts facilitate a deeper understanding of these models, which serve as the cornerstone for more advanced techniques, such as Random Forest.
- **Model Validation**: We will emphasize the importance of model validation. We will discuss different techniques for evaluating and validating the performance of a model, like cross-validation. Additionally, we will explore the concepts of bias and variance in model evaluation.
- **Principal Components Analysis (PCA):** In this final section, we will explore a technique that allows preprocessing the data in such a way that certain statistical relationships between variables in a dataset are eliminated. Among other things, this technique reduces

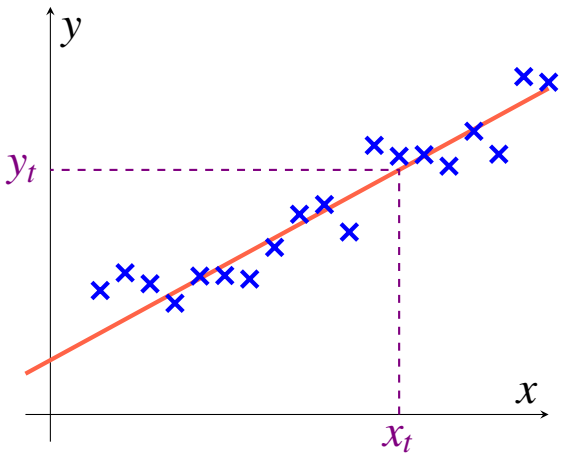
the dimensionality of the data, which can be beneficial in certain scenarios.

Throughout these sections, our goal is to provide an understanding of the different forms that $\hat{S}(x)$ can take, starting from the basic concepts and gradually building up to more advanced techniques. We will also emphasize the significance of model validation to ensure reliable and generalizable trained models.

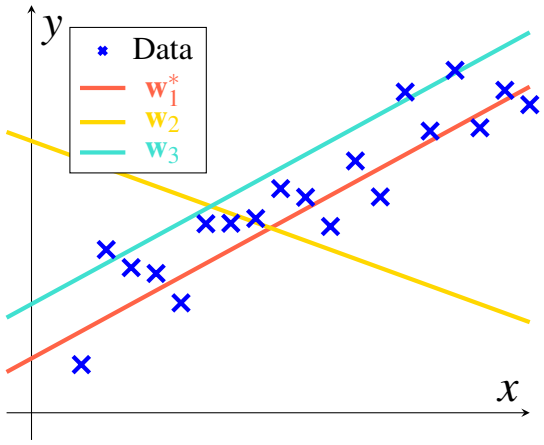
2.2 Linear Regression

Although linear regression has been extensively studied in the field of statistics and, of course, in the areas of data science and artificial intelligence, it serves as a key foundation for understanding techniques of higher complexity. The assumptions made in linear regression extend to these techniques, which is why linear regression is often considered a baseline when comparing performance with other methods.

The first assumption is that there exists a linear (affine) relationship between the data and the labels. Using the notation for n data points $\{x_i, y_i\}_{i=1}^{n}$, the goal is to model a dependency of the form $y = wx + b$. This allows us, as shown in Figure 1a, to predict the value of the label y_t for new values of x_t.



(a) Traditional regression framework for a single-dimensional input.



(b) Possible adjustments of the $\mathbf{w}$, in this case will be the parameters that the model tunes.

Figure 1 – Some representations of linear regression for a function of one variable: (a) Typical representation of a line fitting the data distribution, (b) Tuning of parameters that adjust the line to the data, labeled as $\mathbf{w}_1^*$ since this line has the parameter that best fits the data.

While understanding regression is intuitive when presented in a one-dimensional manner, this understanding can be extended to represent datasets with high dimensionality. In

this way, for example, if the dataset $\mathbf{x} \in \mathbb{R}^2$ is considered [1], it is natural that the model allowing predictions for new data points x_t and their corresponding values of y_t will no longer be a line. Instead, it can be inferred that it will be a plane (Figure 2). If we extend the problem to n dimensions, we would be referring to a hyperplane.

Now, taking the one-dimensional model of regression as $y = wx + b$, we can see that similarly in two dimensions, we would have $y = w_1x_1 + w_2x_2 + b$. This can be generalized to $y = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$, which can be represented as $y = \mathbf{w}^\top \mathbf{x} + b$. Additionally, an additional adjustment can be made by adopting the convention that $x_0 = 1$ and $w_0 = b$, simplifying the equation to $y = \mathbf{w}^\top \mathbf{x}$.

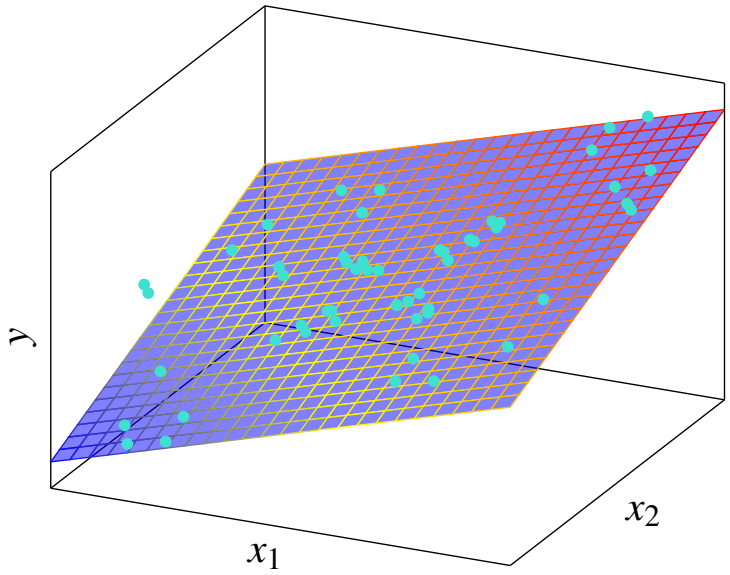


Figure 2 – View of the model fitting a linear regression in two dimensions.

If we adopt the convention $y = \mathbf{w}^\top \mathbf{x}$, then the task of linear regression is to adjust those $\mathbf{w}$ in such a way that the dependency between $\mathbf{x}$ and y is properly modeled. To achieve this, $\mathbf{w}$ should correspond to a model that fits the data $\{x_i, y_i\}_{i=1}^n$ well. To illustrate the it, Figure 1b shows that the choice of $\mathbf{w}$ is crucial in determining how well the data fits the model. Therefore, it is highlighted that $\mathbf{w}_1^*$ is the model that best fits the data.

To build these concepts, we start with a widely used mathematical element, the Euclidean distance, which can be used because we are in the Euclidean space $\mathbb{R}^n$. Therefore, it makes sense for the error function mentioned in the previous section to relate to the distance between the model's prediction and the original data. Here, we can use the one-dimensional representation of regression to illustrate what this distance represents (Figure 3).

[1] It should be noted that the dimensionality of the predictions will never change; we are referring to the fact that we always have continuous values, or equivalently, $y \in \mathbb{R}$

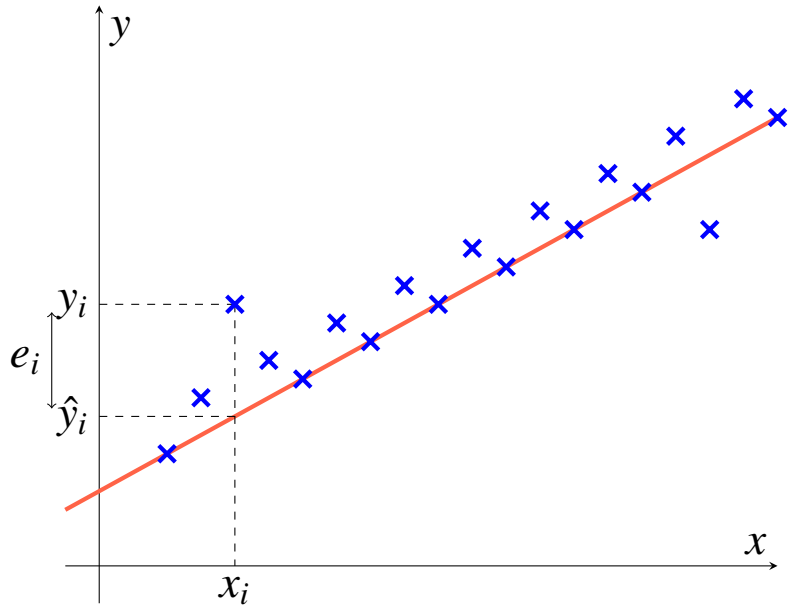


Figure 3 – Graphical representation of error in linear regression, where e_i represents the error between the model's prediction for the i-th data point ($\hat{y}_i$) and the corresponding value y_i for the i-th data point.

Examining the error function, the error as a function of the adjustment of $\mathbf{w}$, we have that:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \mathbf{w}^\top \mathbf{x}_i\right)^2, \tag{2.3}$$

this generates the following optimization problem:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

Now, starting from the error function, we have that:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \mathbf{w}^\top \mathbf{x}_i\right)^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}\left(y_i^2 - 2y_i\mathbf{w}^\top \mathbf{x}_i + \left(\mathbf{w}^\top \mathbf{x}_i\right)^2\right) \\ &= \frac{1}{2}\sum_{i=1}^{n} y_i^2 - \mathbf{w}^\top \mathbf{x}_i y_i + \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}_i\right)\left(\mathbf{w}^\top \mathbf{x}_i\right) \\ &= \underbrace{\frac{1}{2}\mathbf{w}^\top\left(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top\right)}_{\text{QF}}\mathbf{w} - \underbrace{\left(\sum_{i=1}^{n}\mathbf{x}_i y_i\right)^\top}_{\mathbf{b}^\top}\mathbf{w} + \underbrace{\frac{1}{2}\sum_{i=1}^{n} y_i^2}_{c}, \end{aligned} \tag{2.4}$$

where:

$$\mathbf{H} = \sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top, \quad \mathbf{b} = \sum_{i=1}^{n}\mathbf{x}_i y_i, \quad c = \frac{1}{2}\sum_{i=1}^{n} y_i^2$$

From the previous deduction, a particular aspect is highlighted, the first term of the expression, is a quadratic form (QF). This means that $E(\mathbf{w})$ is a quadratic function of $\mathbf{w}$. This

represents an advantage in terms of the optimization problem because it allows the application of an algorithm to search for the optimal $\mathbf{w}^*$ based on convex optimization. Even better, it is possible to find an analytical solution to the above expression as follows:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \nabla_{\mathbf{w}} \left(\frac{1}{2} \mathbf{w}^\top \mathbf{H} \mathbf{w} - \mathbf{b}^\top \mathbf{w} + c \right) = \mathbf{H}\mathbf{w} - b$$

$$\nabla^2_{\mathbf{w}} E(\mathbf{w}) = \mathbf{H}.$$

Note that for any vector $\mathbf{z} \in \mathbb{R}^{d+1}$,

$$\mathbf{z}^\top \mathbf{H} \mathbf{z} = \sum_{i=1}^{n} \left(\mathbf{z}^\top \mathbf{x}_i \right) \left(\mathbf{x}_i^\top \mathbf{z} \right) = \sum_{i=1}^{n} \left(\mathbf{z}^\top \mathbf{x}_i \right)^2, \tag{2.5}$$

that is, $\mathbf{H}$ is positive semidefinite, then the error function $E(\mathbf{w})$ is convex.

If $\mathbf{H}$ is positive definite, then $E(\mathbf{w})$ has a unique global minimum:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = 0 \Longrightarrow \mathbf{w}^* = \mathbf{H}^{-1}\mathbf{b}$$

Having this analytical solution allows us to incorporate the found solution into an iterative algorithm that iteratively finds the value of $\mathbf{w}^*$. In general, this is known as the iterative descent algorithm.

Algoritmo 1: Iterative Descent Algorithm

```
Initialize w_0
repeat
    choose the direction of d
    w_{k+1} = w_k + η_k d
until Any termination condition.
```

It is important to clarify what $\mathbf{d}$ and η_k represent in Algorithm 1. Regarding $\mathbf{d}$, it refers to the descent direction, while η_k represents the learning rate. Although both elements are crucial for the algorithm's operation, it is more beneficial to understand the concept of the descent direction. Therefore, a brief study on this descent direction will be presented next.

Assume that for the choice of $\mathbf{d}$ we have a function of a fixed point $\mathbf{w}$ along a certain direction $\mathbf{d}$.

$$g(\eta) = E(w + \eta \mathbf{d}).$$

From the directional derivative:

$$g'(\eta) = \mathbf{d}^\top \nabla_{\mathbf{w}} E(\mathbf{w} + \eta \mathbf{d}) \Longrightarrow g'(0) = \mathbf{d}^\top \nabla_{\mathbf{w}} E(\mathbf{w}) = \langle \mathbf{d}, \nabla_{\mathbf{w}} E(\mathbf{w}) \rangle \tag{2.6}$$

For $\|\mathbf{d}\|$ constant, the directional derivative is maximally negative when $\mathbf{d} = -\nabla_{\mathbf{w}} E(\mathbf{w})$.

This means that the negative gradient is the direction of the steepest descent. This modification changes the update equation of Algorithm 1 from $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta_k \mathbf{d}$ to $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \nabla \mathbf{w} E(\mathbf{w_k})$.

So far, it has been shown that the error function has a quadratic form, which, if positive definite, defines a convex surface. In the simplest case, it is a parabola, and in higher dimensions, it is a hyperparaboloid. Based on this fact, an analytical solution was found to determine the optimal value $\mathbf{w}^*$ that minimizes this error function. Later on, the translation of this into computational implementation was discussed[2].

Another way to approach linear regression is the probabilistic derivation. It starts with the following assumptions:

- y_i and x_i are related by:

$$y_i = \check{\mathbf{w}}^\top \mathbf{x}_i + \varepsilon_i \tag{2.7}$$

- $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$, and are independent.

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right) \tag{2.8}$$

If y_i is a random variable with density:

$$p\left(y_i | x_i; \check{\mathbf{w}}\right) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\left(y_i - \check{\mathbf{w}}^\top \mathbf{x}_i\right)^2}{2\sigma^2}\right)$$

where $\check{\mathbf{w}}$ is a parameter of the density. Thus, the likelihood function is: (2.9)

$$L\left(\check{\mathbf{w}}\right) = p\left(\mathbf{y} | \mathbf{X}; \check{\mathbf{w}}\right) \xrightarrow{\varepsilon_i \text{ is i.i.d}} L\left(\check{\mathbf{w}}\right) = \prod_{i=1}^{n} p\left(y_i | x_i; \check{\mathbf{w}}\right) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\left(y_i - \check{\mathbf{w}}^\top \mathbf{x}_i\right)^2}{2\sigma^2}\right)$$

From the above, a question arises regarding the choice of $\check{\mathbf{w}}$. What would be a good $\check{\mathbf{w}}$ in this probabilistic case? The immediate answer lies in the principle of maximum likelihood. The idea is to find a $\check{\mathbf{w}}$ that maximizes the probability of the data. In this case, we want to maximize the logarithmic likelihood.

$$\begin{aligned}
\log\left(L\left(\check{\mathbf{w}}\right)\right) &= \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\left(y_i - \check{\mathbf{w}}^\top \mathbf{x}_i\right)^2}{2\sigma^2}\right)\right) \\
&= \sum_{i=1}^{n} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{\left(y_i - \check{\mathbf{w}}^\top \mathbf{x}_i\right)^2}{2\sigma^2}\right)\right)\right) \\
&= n \log \frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{\sigma^2} \underbrace{\frac{1}{2} \sum_{i=1}^{n} \left(y_i - \check{\mathbf{w}}^\top \mathbf{x}_i\right)^2}_{\textbf{MSE!}}
\end{aligned} \tag{2.10}$$

[2] There are other ways to analyze linear regression, for example, it can be examined from the perspective of normal equations (HASTIE *et al.*, 2009).

Under the aforementioned assumptions, maximizing likelihood is equivalent to minimizing the squared error in the data. However, there is an additional study to be done based on this probabilistic derivation. Let's once again decompose the error. Given $(\mathbf{x}, y) \sim D$, let's assume that h is a function of $\mathbf{x}$ with parameters $\mathbf{w}$.

$$\check{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(h(\mathbf{x}_i,\mathbf{w}) - y_i\right)^2$$

In the limit as $n \to \infty$

$$E(\mathbf{w}) = \lim_{n\to\infty}\frac{\check{E}(\mathbf{w})}{n} = \frac{1}{2}\iint\left(h(\mathbf{x},\mathbf{w}) - y\right)^2 p(\mathbf{x},y)dyd\mathbf{x} = \frac{1}{2}\iint\left(h(\mathbf{x},\mathbf{w}) - y\right)^2 p(y|\mathbf{x})p(\mathbf{x})dyd\mathbf{x} \tag{2.11}$$

By performing certain manipulations[3] and integrating over y:

$$\begin{aligned}
&\iint\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2 p(y|\mathbf{x})p(\mathbf{x})dyd\mathbf{x} = \int\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2 p(\mathbf{x})d\mathbf{x}\\
&\int\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)\left(\mathbb{E}\left[y|\mathbf{x}\right] - y\right)p(y|\mathbf{x})dy = 0\\
&\int\left(\mathbb{E}\left[y|\mathbf{x}\right] - y\right)p(y|\mathbf{x})dy = \mathbb{E}\left[y^2|\mathbf{x}\right] - \mathbb{E}\left[y|\mathbf{x}\right]^2
\end{aligned} \tag{2.12}$$

Putting it all together:

$$E(\mathbf{w}) = \frac{1}{2}\int\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2 p(\mathbf{x})d\mathbf{x} + \frac{1}{2}\int\mathbb{E}\left[y^2|\mathbf{x}\right] - \mathbb{E}\left[y|\mathbf{x}\right]^2 p(\mathbf{x})d\mathbf{x},$$

notice that the second integral does not depend on $\mathbf{w}$, and this part of the error function is known as the irreducible error. Another element that becomes evident is that the value of $h(\mathbf{x},\mathbf{w})$ that minimizes $E(\mathbf{w})$ is $\mathbb{E}\left[y|\mathbf{x}\right]$, and from now on, this function will be referred to as the regression function.

To conclude this section, we introduce the bias-variance tradeoff, which encompasses the two remaining components of the error, and some tips to understand preprocessing. It is important to note that these two elements are strongly interrelated, and there exists a compromise between them. The following considerations are taken into account:

In practice, n is finite. Consider datasets with n data points $D_1, D_2, \cdots \sim p(\mathbf{x},y)$. For a fixed $\mathbf{x}$, the term $\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2$ depends on the particular D_i. Thus, averaging over all possible D:

$$\begin{aligned}
\mathbb{E}\left[E(\mathbf{x})\right] &= \mathbb{E}_D\left(h(\mathbf{x},\mathbf{w}) + \mathbb{E}\left[y|\mathbf{x}\right]\right)^2 = \mathbb{E}_D\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}_D h(\mathbf{x},\mathbf{w}) + \mathbb{E}_D h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2\\
&= \mathbb{E}\left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}_D\left(h(\mathbf{x},\mathbf{w})\right)\right)^2 + \left(\mathbb{E}_D\left(h(\mathbf{x},\mathbf{w})\right) - \mathbb{E}\left[y|\mathbf{x}\right]\right)^2
\end{aligned} \tag{2.13}$$

[3] $\left(h(\mathbf{x},\mathbf{w}) - y\right)^2 = \left(h(\mathbf{x},\mathbf{w}) - \mathbb{E}\left[y|\mathbf{x}\right] + \mathbb{E}\left[y|\mathbf{x}\right] - y\right)^2 = \left(h(\mathbf{x},\mathbf{w}) + \mathbb{E}\left[y|\mathbf{x}\right]\right)^2$

From the above, it can be said that $(\mathbb{E}_D(h(\mathbf{x},\mathbf{w})) - \mathbb{E}[y|\mathbf{x}])^2$ is the bias of h at $\mathbf{x}$, and $\mathbb{E}(h(\mathbf{x},\mathbf{w}) - \mathbb{E}_D(h(\mathbf{x},\mathbf{w})))^2$ is the variance of h at $\mathbf{x}$. Integrating over $\mathbf{x}$:

$$\begin{aligned} \text{bias}^2 &= \frac{1}{2}\int (\mathbb{E}_D(h(\mathbf{x},\mathbf{w})) - \mathbb{E}[y|\mathbf{x}])^2 p(\mathbf{x})d\mathbf{x} \\ \text{variance} &= \frac{1}{2}\int \mathbb{E}(h(\mathbf{x},\mathbf{w}) - \mathbb{E}_D(h(\mathbf{x},\mathbf{w})))^2 p(\mathbf{x})d\mathbf{x} \end{aligned} \tag{2.14}$$

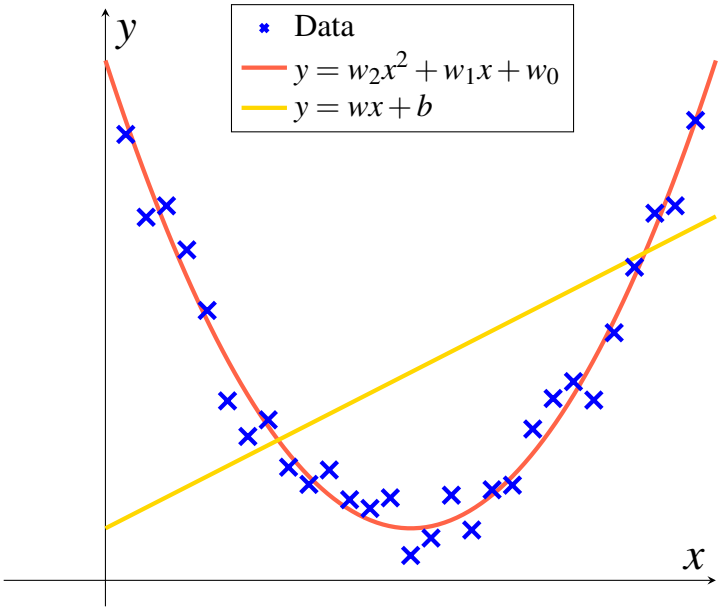


Figure 4 – Choosing the appropriate model according to the nature of the data is crucial. We demonstrate the fitting of data using a quadratic curve and a straight line.

Let's consider an example of a dataset that doesn't visually follow a straight line, as shown in Figures 1 and 3. Suppose the data now follows a parabolic pattern as depicted in Figure 4. This might lead to confusion as it is no longer a linear function that fits the data. However, careful examination and appropriate model selection can address this issue. For the data distribution in Figure 4, the best choice would be $y = w_2x^2 + w_1x + w_0$. A simple inspection reveals that the nature of the model remains linear in terms of the parameters. When comparing the fit with the linear model, it becomes evident that the linear model is not the optimal choice. To have a clearer understanding, linearity applies to the parameters of the model and not necessarily to the nature of the data, as long as a strategy is found to transform the problem into the usual linear model. For instance, let's examine the previous case, which was one-dimensional, and

extend it to two dimensions as follows:

$$y = w_{11}x_1^2 + w_{22}x_2^2 + w_{12}x_1x_2 + w_1x1 + w_2x_2 + w_0$$

For d dimensions:

$$\begin{aligned} y =& w_{11}x_1^2 + w_{22}x_2^2 + \cdots + w_{dd}x_d^2 \\ &+ w_{12}x_1x_2 + w_{13}x_1x_3 + \cdots + w_{23}x_2x_3 + \cdots + w_{(d-1)d}x_{d-1}x_d \\ &+ w_1x_1 + w_2x_2 + \ldots w_dx_d + w_0 \\ &\binom{d+2}{2} \quad \text{terms.} \end{aligned} \tag{2.15}$$

This means that the combination shown in the previous equation determines the number of parameters to be adjusted in the model. In the case of two dimensions, it would be a combination of four choose two, resulting in a total of six parameters to be adjusted: $\{w_{11}, w_{22}, w_{12}, w_1, w_2, w_0\}$.

2.3 K-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm is a versatile and intuitive method widely employed for both classification and regression tasks in machine learning. The k-NN algorithm makes minimal assumptions about the underlying structure of the data. This flexibility enables the k-NN method to adapt effectively to a wide variety of situations, constructing a decision boundary that can conform to irregular patterns in the data. However, this adaptability comes at a cost. The decision boundary produced by the k-NN algorithm can be highly sensitive to fluctuations in the training data. Each subregion of the decision boundary is determined by a small subset of the training instances. Consequently, the exact position of these instances can cause the boundary to be irregular and unstable.

The subsequent sections delve deeper into the workings of the k-NN algorithm, its strengths and weaknesses, and the trade-offs involved in its use.

2.3.1 Problem Setup

Consider a training dataset $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ is a feature vector and $y_i \in Y$ is the corresponding label. For classification, Y is a discrete set, while for regression, Y is a continuous set.

2.3.2 Distance Measure

The distance function $d(x,x')$ can vary according to the specific problem and feature type. For continuous features, the Euclidean distance is often used, defined for two vectors $x,x' \in \mathbb{R}^d$ as $d(x,x') = \sqrt{\sum_{i=1}^{d}(x_i - x'_i)^2}$. Other distance measures, such as Manhattan or Minkowski distance, can also be used.

2.3.3 The Algorithm

Given an unlabelled instance x, the k-NN algorithm operates as Algorithm 2 describes:

Algoritmo 2: K-Nearest Neighbors Algorithm

Input: Unlabelled instance x, training dataset T, number of neighbors k, distance metric d.

Procedure:

Compute $d(x,x_i)$ for each instance $x_i \in T$.
Identify the k instances $N_k(x) = \{x_{(1)},\ldots,x_{(k)}\}$ in T that are nearest to x.
Determine the prediction $\hat{y}$:
if Y is discrete
$\hat{y} = \text{mode}(\{y_{(1)},\ldots,y_{(k)}\})$
else
$\hat{y} = \frac{1}{k}\sum_{i=1}^{k} y_{(i)}$
end if

Output: Predicted label $\hat{y}$.

2.3.4 Theoretical Considerations

The k-NN rule is a classification and regression method predicated on the notion that observations in close proximity within a feature space share similar labels. Given a novel data point, the k-NN rule identifies the k nearest points within the training set, assigning a label based on the majority of those k nearest neighbors' labels. Various metrics, such as Euclidean distance, can be employed to measure the distance between points.

In the other hand, Bayes risk R^* represents a theoretical measure of the minimum error expected from a classifier, given knowledge of the underlying true probability distribution of the data. It is associated with the optimal Bayes decision rule, which classifies an observation into the class that maximizes the posterior probability given the observation. In essence, it is the

lowest achievable error in classification, assuming all conditional probabilities are known. The k-NN decision rule approaches the Bayes risk as the number of instances n tends to infinity, and k also grows but at a slower rate than n (i.e., $\frac{k}{n} \to 0$). This implies that the k-NN rule can be a consistent approximation to the optimal Bayes rule in the limit of an infinite amount of data.

2.3.4.1 Convergence Theorem

Under certain regularity conditions, as the number of instances n goes to infinity and for fixed k, the k-NN decision rule approaches the Bayes rule. The formal statement of the theorem is as follows:

Let R_n be the risk of the k-NN rule and R^* be the Bayes risk. Then, if $k \to \infty$ and $k/n \to 0$ as $n \to \infty$, we have:

$$\lim_{n\to\infty} \mathbb{E}[R_n] = \mathbb{E}[R^*] \tag{2.16}$$

where $\mathbb{E}[R_n]$ and $\mathbb{E}[R^*]$ denote the expected risks.

2.3.4.2 Choice of k

The choice of k balances the bias-variance trade-off. Formally, the expected prediction error of k-NN can be decomposed as follows:

$$E\left[(y-\hat{y})^2\right] = \sigma^2 + \frac{\sigma^2}{k} + Bias^2(\hat{f}(x)), \tag{2.17}$$

where σ^2 is the irreducible error, $1/k$ represents the model's variance, and $Bias^2(\hat{f}(x))$ is the squared bias.

2.3.5 Trade-offs

The k-NN algorithm manifests an array of trade-offs that need to be carefully considered. Key among these is the bias-variance trade-off, which is greatly influenced by the choice of k. A small value of k results in a low bias, high variance scenario, which means the model closely follows the training data but may respond poorly to unseen data. On the contrary, a large value of k gives rise to a model that has high bias but low variance, implying that the model may oversimplify and miss certain complexities in the data.

The k-NN algorithm is also characterized by computational challenges. It must compute distances between instances, an operation that grows more costly with an increasing volume of data. Therefore, as the size of the dataset increases, the algorithm can become significantly slower and less efficient.

Despite these challenges, the k-NN algorithm boasts notable advantages. Its simplicity and intuitiveness are key strengths. There are no assumptions made about the functional form of the data, allowing it to model complex patterns that other algorithms might miss. Additionally, the algorithm can be highly effective if the decision boundary is very irregular, flexibly adapting to the data's structure.

However, the k-NN algorithm also exhibits certain weaknesses. As mentioned earlier, it suffers significantly as the volume of data increases. Another critical trade-off arises from the so-called "curse of dimensionality". k-NN does not handle high-dimensional data effectively due to this curse. The distance metrics used become less meaningful as the number of dimensions grows, leading to a deterioration in the algorithm's performance. Moreover, it is sensitive to irrelevant features and the scale of the data. Each feature contributes equally to the distance calculation, which means the algorithm could perform poorly if the features are not carefully selected and preprocessed.

We have explored two learning techniques for prediction thus far: the stable yet biased linear model, and the less stable but seemingly less biased class of k-nearest-neighbor estimates. With a sufficiently large training dataset, it might appear that we could always approximate the theoretically optimal conditional expectation using k-nearest-neighbor averaging. We should be able to find a reasonably large neighborhood of observations close to any given point x and average them. However, this approach and our intuition break down in high-dimensional spaces, giving rise to the well-known phenomenon referred to as the curse of dimensionality (BELLMAN, 2015). This issue manifests in various ways, which we will now examine.

Let's consider the nearest-neighbor procedure for inputs uniformly distributed within a p-dimensional unit hypercube, as depicted in Figure 5. In this scenario, we define a hypercubical neighborhood around a target point to encompass a fraction r of the observations. This fraction corresponds to a fraction r of the unit volume, leading to an expected edge length of the hypercube represented by $e_p(r) = r^{1/p}$. For instance, in ten dimensions, we have $e_{10}(0.01) = 0.63$ and $e_{10}(0.1) = 0.80$, while the entire range for each input is only 1.0. Therefore, to capture 1% or 10% of the data and compute a local average, we need to cover 63% or 80% of the range for each

input variable. It becomes evident that these neighborhoods can no longer be considered truly "local." Additionally, reducing the value of r dramatically does not provide significant benefits, as averaging fewer observations results in a higher variance in our fit.

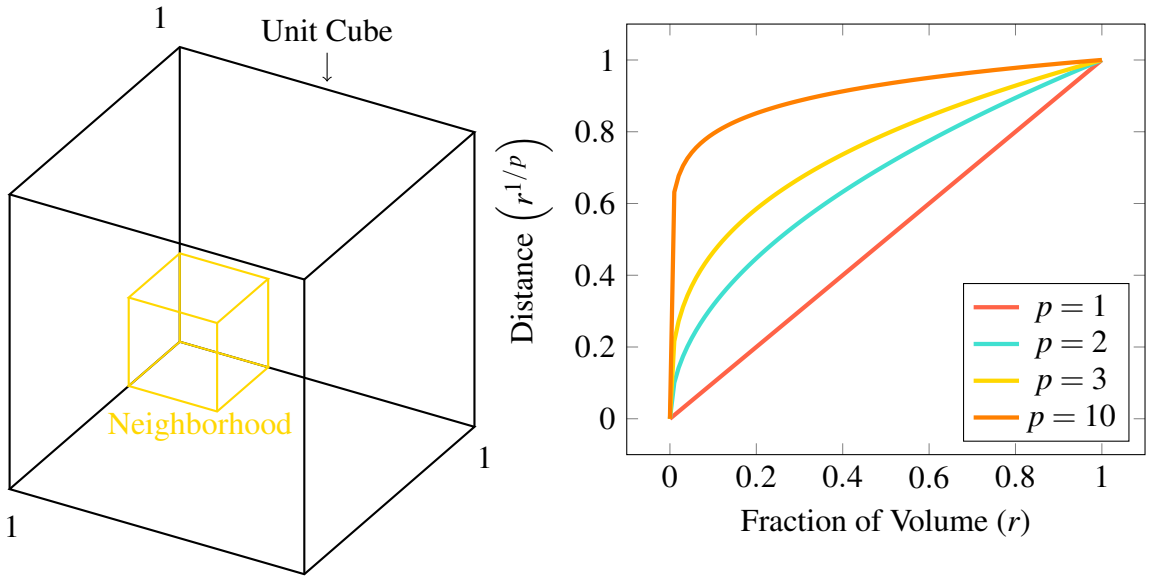


Figure 5 – The Curse of Dimensionality with a Subcubical Neighborhood for Uniform Data in a Unit Cube. The figure on the right demonstrates the side-length of the subcube necessary to capture a fraction r of the data's volume, considering different dimensions p. Notably, in the case of ten dimensions, it becomes crucial to cover 80% of the range for each coordinate to capture just 10% of the data

2.4 Boosting

To commence this section and gradually develop the concept of Boosting, we delve into fundamental concepts such as classifier combination. The essence of this technique revolves around classifiers, as it was initially devised for addressing classification problems. Nevertheless, its applicability can be readily extended to regression problems.

The idea is to construct a classifier that is a combination of multiple classifiers. This combined classifier is defined by obtaining classifiers $h_1, h_2, \ldots, h_T$ that minimize the error on different versions of the data[4].

$$f(\mathbf{x}) = \sum_{i=1}^{T} \alpha_i h_i(\mathbf{x}) \tag{2.18}$$

[4] $f(\mathbf{x}) = \sum_{i=1}^{T} \alpha_i h_i(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + \cdots + \alpha_T h_T(\mathbf{x})$, this means that the equation 2.18 is a linear combination of classifiers $h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x})$, α_i can be restricted depending on the learning problem and taking into account if certain objectives or properties are to be achieved, for example, if $\sum_{i=1}^{T} \alpha_i = 1$, the combination would no longer be linear, it would become convex.

Classifying using the sign of f:

$$h(\mathbf{x}) = \text{sign}\,(f(x)) = \begin{cases} 1 & \text{if } f(\mathbf{x}) \geq 0, \\ 0 & \text{if } f(\mathbf{x}) < 0 \end{cases} \tag{2.19}$$

The inspiration behind boosting arose from the desire to create a technique that merges the predictions of multiple "weak" classifiers to form a formidable and influential "committee". To introduce the notions of Boosting, it is stated that within this context, the objective of Probably Approximately Correct Learning (PAC Learning) is to find a hypothesis h that satisfies the condition $\mathbf{P}(e(h) \geq \varepsilon) \leq \delta$. In simpler terms, the goal is to ensure that the probability of the hypothesis h having a generalization error greater than a specified tolerance level, denoted by ε, is less than a certain confidence level, denoted by δ. The aim is to have a high level of confidence (with a confidence level of $1-\delta$) that the generalization error will be lower than ε. Suppose there exists an h_d that satisfies the condition of weak learnability, $\mathbf{P}(e(h_d) \geq \varepsilon_0) \leq \delta_0$, for any data distribution. This raises the question of whether it is possible to achieve strong learnability from weak learnability. The answer, for now, is yes! (SCHAPIRE, 1990) However, this topic will be further explored and discussed in the development of this section.

The explanation begins by introducing the simplest boosting algorithm to develop an initial intuition and gradually demonstrate its usefulness. It is assumed that there exists an algorithm, denoted as A, which takes data from a distribution D as input, i.e., $\{x_i, y_i\} \sim D$, and produces a model h with an error bound of $e(h) \leq \beta$, regardless of the distribution D. Building upon this, a modest algorithm is proposed that involves requesting data (x_i, y_i) from an oracle, denoted as $EX(c, D)$. The algorithm proceeds as follows:

1) $h_1 \leftarrow A(D_1)$ with $D_1 = D$
2) $h_2 \leftarrow A(D_2)$ where D_2 is such that $\mathbf{P}_{D_2}[h_1(x) \neq c(x)] = \frac{1}{2}$
3) $h_3 \leftarrow A(D_3)$ where D_3 is such that $\mathbf{P}_{D_3}[h_1(x) \neq h_2(x)] = 1$

   returns $h(x)$ = majority $(h_1(x), h_2(x), h_3(x))$

It can be shown that:

$$\left.\begin{array}{l} \mathbf{P}_D[h_1(x) \neq c(x)] \leq \beta \\ \mathbf{P}_{D_2}[h_2(x) \neq c(x)] \leq \beta \\ \mathbf{P}_{D_3}[h_3(x) \neq c(x)] \leq \beta \end{array}\right\} \Longrightarrow \mathbf{P}_D[h(x) \neq c(x)] \leq 3\beta^2 - 2\beta^3 \tag{2.20}$$

This means that the error generated by the model is bounded by a function that depends on β, as shown in Figure 6. In terms of error, it signifies a significant reduction

compared to the original error. It is important to note that the entire procedure begins with a hypothesis that allows the model to intuitively learn a weak advantage expressed in the original data distribution, D.

An intuitive next step is to view the aforementioned procedure as a subroutine, utilizing it recursively to enhance the accuracy of weaker hypotheses. Assuming that A returns an h with $e(h) \leq \beta < \frac{1}{2}$ for any distribution D, the natural goal is to achieve $e(h) \leq \varepsilon$. If the weak learning algorithm guarantees $e(h) \leq g^{-1}(\varepsilon) > \varepsilon$, then the previous modest algorithm is applied. In case these guarantees are not met, a natural option is to recursively apply the modest algorithm as mentioned earlier.

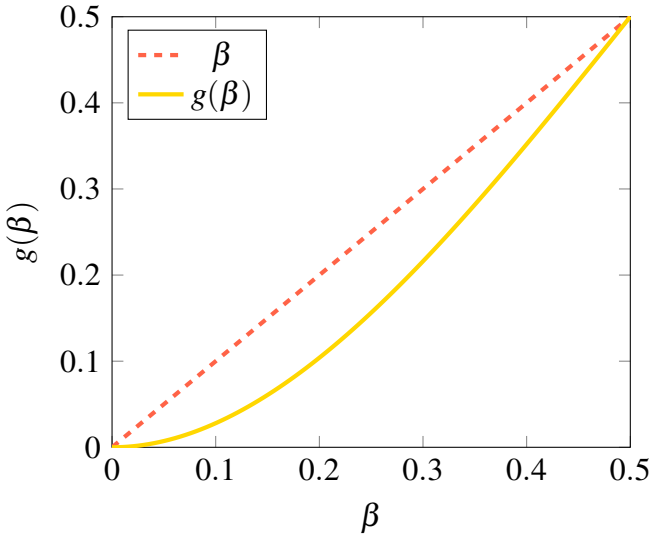


Figure 6 – Error Bound of Hypothesis h in Boosting Algorithm A. The graph illustrates the error bound of hypothesis h, generated by the A algorithm, which is constrained by the function $g(\beta) = 3\beta^2 - 2\beta^3$. This function showcases a significantly lower error compared to the original error.

Algoritmo 3: A strong boosting algorithm (ε, D')

if $\varepsilon \geq \varepsilon_{WL}$ **then**
  Returns $DT(\varepsilon, D')$
end if
$\beta \leftarrow g^{-1}(\varepsilon)$
$h_1 \leftarrow \text{StrongBoosting}(\beta, D'_1)$
$h_2 \leftarrow \text{StrongBoosting}(\beta, D'_2)$
$h_3 \leftarrow \text{StrongBoosting}(\beta, D'_3)$
$h \leftarrow \text{majority}(h_1, h_2, h_3)$
returns h

By examining the structure of Algorithm 3, it becomes apparent that the recursion

employed in the algorithm increases the depth of the tree, raising concerns about its efficiency (Figure 7). Another aspect that can contribute to the study of the algorithm's efficiency is the number of calls made to the oracle $EX(c,D)$.

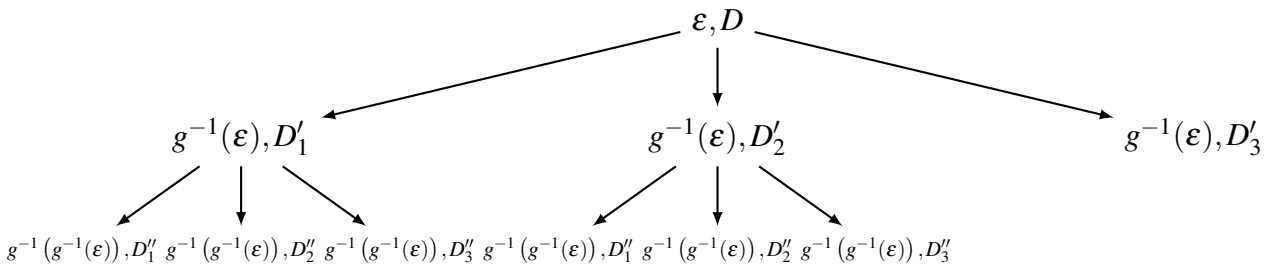


Figure 7 – Recursion Tree Depth. In the first level of the tree, we illustrate the initial recursion call. However, this can easily be extended to a second level where two recursive calls are made.

Up to this point, it can be stated that the boosting algorithm in the PAC model requires constant queries to the oracle, generates a non-regular structure, relies on knowing a guarantee of error from the weak algorithm, and overall is not practical enough due to limitations arising from its poor efficiency.

Although Boosting in the PAC model may not be practical, it serves as an introduction to a methodology with new considerations for dealing with the effects of this impracticality. We start with a dataset $S = \{\mathbf{x}i, y_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathscr{X}$ and $y_i \in \{-1,1\}$. The data is associated with a weight vector $D = \{D_1, D_2, \ldots, D_n\}$, which forms a distribution, i.e.,

$$D_i \geq 0 \quad \text{y} \quad \sum_{i=1}^{n} D_i = 1. \tag{2.21}$$

There exists a class of base hypotheses $h \in \mathscr{H}$ where $h : \mathscr{X} \to \{-1,1\}$, which allows us to calculate the weighted error of a hypothesis h according to D as follows:

$$e_D(h) = \sum_{i=1}^{n} D_i I_{\{y_i f(\mathbf{x}_i) \leq 0\}} = \sum_{i:h(\mathbf{x}_i) \neq y_i} D_i \tag{2.22}$$

Assuming we have access to a weak learner A, which takes S and D as inputs and returns $h \in H$ with $e_D(h) < \frac{1}{2}$, AdaBoost proceeds through a series of rounds $1,2,\ldots$, where it obtains hypotheses $h_1, h_2, \ldots$. In the first round, the weak learner (A) is called with the uniform distribution $D_i = \frac{1}{n}$, where $i = 1,2,\ldots,n$. In the next round, D is modified such that D_i increases if $h_1(\mathbf{x}) \neq y_i$, and decreases when $h_1(\mathbf{x}) = y_i$. This procedure is iterated, modifying the weights in each round according to the hypothesis from the previous round. Now, we construct $f(\mathbf{x})$ to minimize:

$$e(f) = \frac{1}{n} \sum_{i=1}^{n} e^{-y_i f(\mathbf{x}_i)} \geq \frac{1}{n} \sum_{i=1}^{n} I_{\{y_i f(\mathbf{x}_i) \leq 0\}} \tag{2.23}$$

Minimizing the margin cost function on the data. To achieve this, we assume that α_j and h_j are known for $j = 1, 2, \ldots, k-1$, and we aim to find α_k and h_k. Let $f_k = \sum_{j=1}^{k} \alpha_j h_j$ be denoted as the sum of the previous hypotheses.

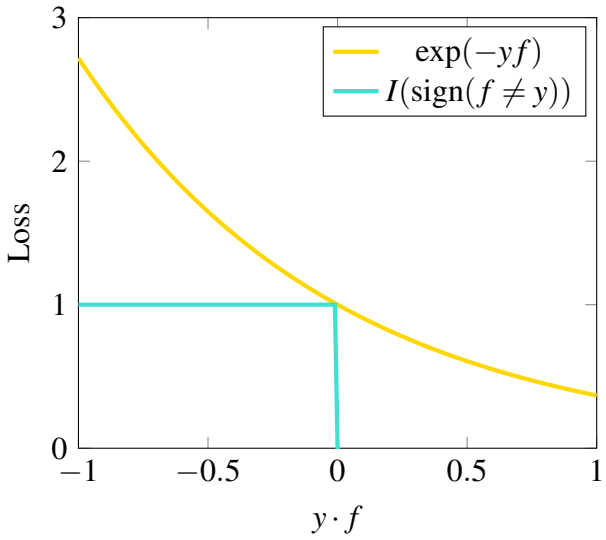


Figure 8 – Loss functions for binary classification tasks. The target variable is represented as $y = \pm 1$, and the prediction is denoted as f, where the class prediction is determined by the sign of f. The two loss functions considered are misclassification, defined as an indicator function when the sign of f is not equal to y, and exponential loss, given by the exponential of the negative product of y and f.

$$
\begin{aligned}
e(f_k) &= \frac{1}{n} \sum_{i=1}^{n} e^{-y_i f(\mathbf{x}_i)} \\
&= \frac{1}{n} \sum_{i=1}^{n} e^{-y_i f_{k-1}(\mathbf{x}_i) - y_i \alpha_k h_k(\mathbf{x}_i)} \\
&= \frac{1}{n} \left(\sum_{i=1}^{n} e^{-y_i f_{k-1}(\mathbf{x}_i)} \right) \sum_{i=1}^{n} \underbrace{\left(\frac{e^{-y_i f_{k-1}(\mathbf{x}_i)}}{\sum_{i=1}^{n} e^{-y_i f_{k-1}(\mathbf{x}_i)}} \right)}_{D_i} e^{-y_i \alpha_k h_k(\mathbf{x}_i)}
\end{aligned}
\tag{2.24}
$$

From this point onward, the objective is to find α and h that minimize:

$$
\begin{aligned}
\sum_{i=1}^{n} D_i e^{-y_i \alpha h \mathbf{x}_i)} &= \sum_{i:h(\mathbf{x}_i) \neq y_i} D_i e^{\alpha} + \sum_{i:h(\mathbf{x}_i) = y_i} D_i e^{-\alpha} \\
&= e_D(h) e^{\alpha} + (1 - e_D(h)) e^{-\alpha}
\end{aligned}
\tag{2.25}
$$

Therefore, the optimization problem is formulated as follows:

$h = \arg\min_{g \in \mathscr{H}} e_D(g) \Longrightarrow$ weak learner

With h fixed, we find α by taking the derivative and setting it equal to zero: (2.26)

$$\alpha = \frac{1}{2}\ln\left(\frac{1-e_D}{e_D}\right)$$

Algoritmo 4: AdaBoost

$D_1(i) = 1/n$ para $i = 1 \ldots n$
for $t = 1$ to T **do**
  $h_t \leftarrow A(S, D_t)$
  $\varepsilon_t = \sum_{i:h_t(X_i) \neq y_i} D_t(i)$
  $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
  Update D : $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
  Where Z_t normalize D such that $\sum_{i=1}^{t} D_{t+1}(i) = 1$
end for
Returns $f(x) = \sum_{i=1}^{T} \alpha_t h_t(x)$

Note that in Algorithm 4

$$D_{t+1}(i) = \begin{cases} \frac{D_t(i)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}}}{Z_t} & \text{if } y = h_t(\mathbf{x}_i), \\ \frac{D_t(i)\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}}{Z_t} & \text{if } y \neq h_t(\mathbf{x}_i). \end{cases} \tag{2.27}$$

The weighted error of h_t with respect to D_{t+1} is:

$$\begin{aligned} e_{D_{t+1}}(h) &= \sum_{i:h(\mathbf{x}_i) \neq y_i} D_{t+1}(i) \\ &= \sum_{i:h(\mathbf{x}_i) \neq y_i} \frac{D_t(i)\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}}{Z_t} \\ &= \frac{\sqrt{\varepsilon_t(1-\varepsilon_t)}}{\sqrt{\varepsilon_t(1-\varepsilon_t)} + \sqrt{\varepsilon_t(1-\varepsilon_t)}} = \frac{1}{2} \end{aligned} \tag{2.28}$$

Now, the empiric error:

$$\begin{aligned} &\frac{1}{n}\sum_{i=1}^{n} I_{\{y_i f(\mathbf{x}_i) \leq 0\}} \leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(\mathbf{x}_i)} \quad \text{and} \quad D_{t+1}(i) = \frac{e^{-\sum_t \alpha_t y_i h_t(\mathbf{x}_i)}}{n\prod_t Z_t} = \frac{e^{-y_i f(\mathbf{x}_i)}}{n\prod_t Z_t} \\ &\frac{1}{n}\sum_{i=1}^{n} I_{\{y_i f(\mathbf{x}_i) \leq 0\}} \leq \prod_t Z_t = \prod_t 2\sqrt{\varepsilon_t(1-\varepsilon_t)} \end{aligned} \tag{2.29}$$

From the previous deduction, it can be observed that if $\varepsilon_t < \frac{1}{2}$, the empirical error decreases exponentially! Furthermore, if $\varepsilon_t < \frac{1}{2}$, the empirical error reaches zero in a finite

number of steps. There are implications arising from this convergence to the error, with one notable aspect being overfitting of the training data, which can eventually occur for significantly large values of T. Furthermore, under certain considerations, Algorithm 4 can be adopted to classify multiple categories or labels, for example, by using the Hamming loss function[5]. Another commonly used modification is to perform multiclass classification using ranking. However, these specific techniques of machine learning will not be further explored in these sections.

2.5 Tree-Based Models

Several straightforward yet commonly used models function by dividing the input space into cuboid regions aligned with the axes and assigning a simple model (such as a constant) to each region. These models can be considered as a method of combining models, where only one model is responsible for making predictions at any specific point in the input space. The procedure of choosing a particular model based on a new input $\mathbf{x}$ can be likened to a sequential decision-making process(BISHOP, 2006).

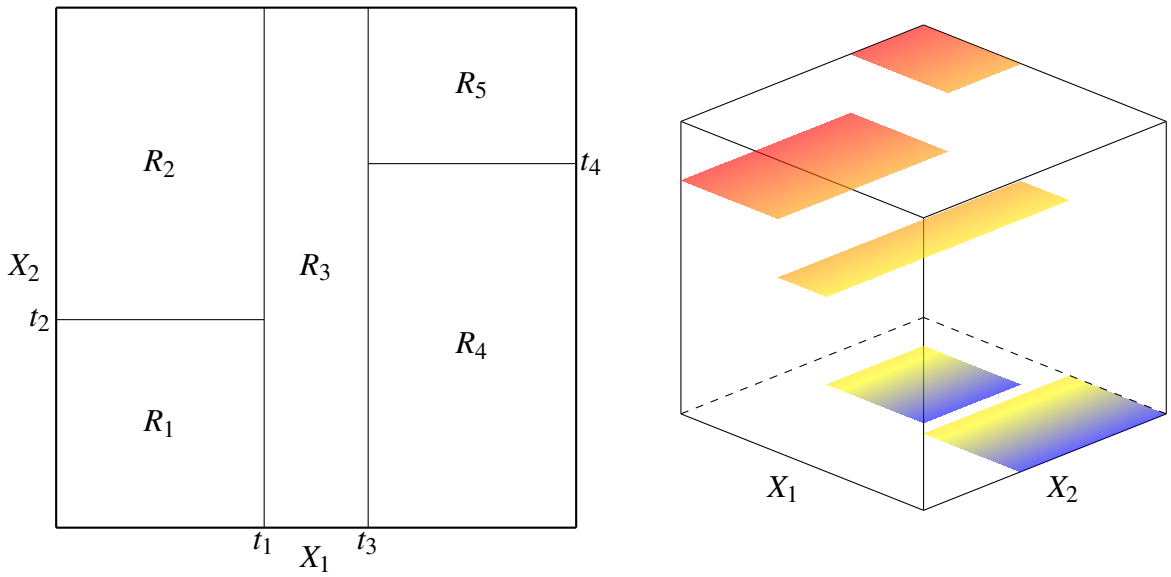


Figure 9 – Fragment space: The right panel displays a partition of a two-dimensional feature space through recursive binary splitting, while a perspective plot of the prediction surface can be seen in the bottom right panel.

Consider a regression problem where the response variable Y is continuous, and the inputs X_1 and X_2 take values within the unit interval. We begin by dividing the space into two regions and model the response by taking the mean of Y within each region. The choice

[5] Based on Hamming distance. The Hamming distance between two vectors is the number of mismatches between corresponding entries.

of variable and split-point is made to achieve the best fit. This process is then repeated for one or both of these regions, leading to further splits, until a stopping rule is applied. In Figure 9, the top right panel demonstrates this process, where we initially split at $X_1 = t_1$. Subsequently, the region $X_1 \leq t_1$ is split at $X_2 = t_2$, and the region $X_1 > t_1$ is split at $X_1 = t_3$. Finally, the region $X_1 > t_3$ is split at $X_2 = t_4$. This results in a partition of the space into five regions denoted as $R_1, R_2, \ldots, R_5$ in the figure. The corresponding regression model predicts the value of Y within each region with a constant value c_m in region R_m, given by:

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{5} c_m I_{\{(X_1,X_2)\in R_m\}}. \tag{2.30}$$

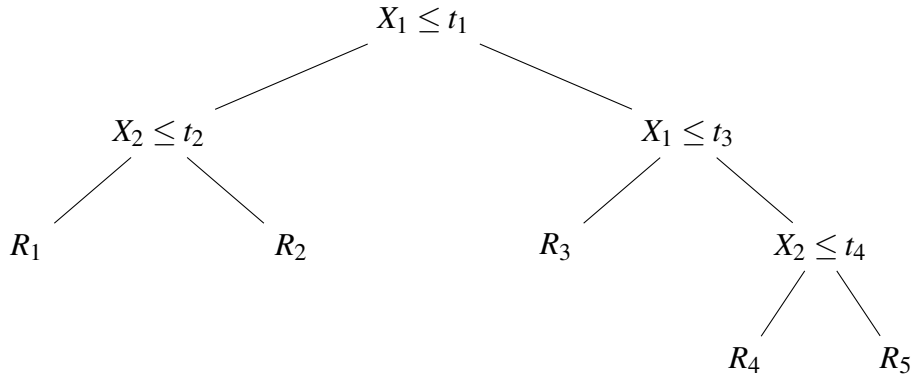


Figure 10 – Decision tree that partitions the feature space (in this case two-dimensional X_1, X_2) by recursive binary splitting.

One significant benefit of the recursive binary tree[6] (Figure 10) is its interpretability. The entire feature space partition can be fully described by a single tree. Although it becomes challenging to visually depict partitions like the one in the top right panel of Figure 9 when there are more than two inputs, the binary tree representation still functions in the same manner.

The process of growing a regression tree is now discussed. The dataset being considered comprises p inputs and a corresponding response for each of N observations, denoted as $(\mathbf{x}_i, y_i)$ for $i = 1, 2, \ldots, N$, where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The algorithm is responsible for making automated decisions regarding the choice of splitting variables, split points, and the overall topology or shape of the tree. Initially, the data is partitioned into M regions, labeled as $R_1, R_2, \ldots, R_M$, and the response is modeled as a constant c_m within each respective region:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m I_{\{\mathbf{x}\in R_m\}}. \tag{2.31}$$

[6] A recursive binary tree is so named due to its inherent property of bifurcating the feature space into two distinct partitions whenever a query is executed. This characteristic is visually depicted in both Figure 9 and Figure 10, which serve as equivalent representations of this structure.

By adopting the criterion of minimizing the sum of squares $\sum (y_i - f(\mathbf{x}_i))^2$, it becomes evident that the optimal estimate for c_m^* is simply the average of y_i within region R_m:

$$c_m^* = \text{ave}\,(y_i | \mathbf{x}_i \in R_m) \tag{2.32}$$

Finding the optimal binary partition in terms of minimizing the sum of squares is often computationally infeasible. Therefore, a greedy algorithm is employed instead. The algorithm starts with all the data and considers a splitting variable j and a split point s. It then defines two half-planes based on this split, namely:

$$R_1(j,s) = \left\{\mathbf{x} | \mathbf{x}_j \leq s\right\} \text{ and } R_2(j,s) = \left\{\mathbf{x} | \mathbf{x}_j > s\right\}$$

Next, the objective is to find the optimal splitting variable j and split point s:

$$\min_{s,j} \left[\min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\mathbf{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right] \tag{2.33}$$

For any chosen splitting variable j and split point s, the inner minimization is solved by:

$$c_1^* = \text{ave}\,(y_i | \mathbf{x}_i \in R_1(j,s)) \text{ and } c_2^* = \text{ave}\,(y_i | \mathbf{x}_i \in R_2(j,s))$$

For each splitting variable, the split point s can be determined quickly by scanning through all of the input data. This enables the identification of the best pair (j,s) for the split. Once the optimal split is found, the data is divided into two resulting regions, and the splitting process is recursively applied to each of these regions. This iterative splitting process is repeated on all of the resulting regions.

2.6 Model Selection & Regularization

In traditional literature (HASTIE *et al.*, 2009; BISHOP, 2006), the topics of regularization and model selection are typically addressed in relation to the specific methods being explained. However, this document takes a deliberate approach by first introducing the ML methods, their theoretical foundations, and the implications of implementing these algorithms. It then proceeds to discuss the objectives of regularization in each of these models. As seen thus far, most of these techniques involve solving an optimization problem that depends on the specific task the algorithm is addressing. At this stage, the idea is to modify the original cost functions in some way and study the consequences of these adjustments, both in terms of model complexity and the reduction of error achieved through optimization.

So, there is a dataset that follows a certain distribution $(x,y) \sim D$, and there exists a class of hypotheses $\mathscr{H}$ (e.g., decision trees, KNN, or any other ML algorithm). The objective is

to find $h \in \mathscr{H}$ that minimizes a chosen error criterion, such as $e(h) = \mathbf{P}_D[h(x) \neq y]$. The aim is to strike a balance between the complexity of $\mathscr{H}$ and the fit of $h \in \mathscr{H}$ to the training data. This balance has several implications:

- If $\mathscr{H}$ is too simple (e.g., a linear regression), it may not provide a good approximation to the underlying function that we want to learn. The model may lack the capacity to capture complex relationships and patterns in the data, leading to high bias and potentially poor predictive performance.
- If $\mathscr{H}$ is too complex (e.g., a random forest with many parameters), it can fit the training data very well and potentially learn the desired function. However, due to its complexity, the model may suffer from overfitting, meaning it becomes too specialized to the training data and fails to generalize well to new, unseen data. This can result in poor performance and unreliable predictions.

The issues mentioned above are particularly critical when the number of available data points is small. In such cases, the model's ability to accurately capture the underlying patterns in the data becomes challenging. Additionally, if the data contains a certain level of noise or variability, the model's performance can be further compromised. Limited data and noise pose significant challenges for complex models, as they can amplify the effects of overfitting and hinder the model's ability to generalize well to new, unseen data.

In model selection algorithms, it is assumed that the complexity of the model class is a variable to be determined by the learning algorithm. For this purpose, consider the nested sequence of hypothesis classes: $\mathscr{H}_1 \subseteq \mathscr{H}_2 \subseteq \cdots \subseteq \mathscr{H}_d \subseteq \ldots$. The model selection process involves two steps. First, a candidate function h_i is selected from each class $\mathscr{H}_i$, typically by minimizing an empirical error criterion within $\mathscr{H}$. Second, a criterion is used to select a function h from the set $\{h_1, h_2, \ldots, h_d, \ldots\}$ such that the error $e(h)$ is minimized. The model selection algorithm aims to strike a balance between the complexity of the model and its ability to fit the training data. By considering a sequence of nested hypothesis classes, the algorithm can explore different levels of complexity and choose the model that achieves the best trade-off between model complexity and training error. This approach allows for flexibility in model selection, as it can adapt to the complexity requirements of the specific problem at hand.

It is possible to directly estimate $e(h_i)$ by considering certain considerations. Given a dataset S, it is divided into subsets S_{train} and S_{test}, where $|S_{train}| = (1-\gamma)|S|$ and $|S_{test}| = \gamma|S|$, with $\gamma \in (0,1)$. Using this division, a candidate hypothesis $h_d \in \mathscr{H}_d$ is found by minimizing

the empirical error (or surrogate loss) on S_{train}. The candidate hypothesis h_d with the lowest empirical error on S_{test} is selected:

$$h_d^* = \underset{\{h_1,h_2,\ldots,\}}{\arg\min}\ \hat{e}_{S_{test}}(h_d) \tag{2.34}$$

Using Chernoff bounds[7,8], it is known that in order to make a probabilistic statement about $e(h)$ with precision ε and confidence $1-\delta$, the following requirements are needed:

$$|S_{test}| \geq \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}. \tag{2.35}$$

However, in practice, it may not always be feasible to have a sufficient number of samples to provide a reliable estimation based on the analysis of the bound. Therefore, the selection of γ becomes an important aspect to consider. If γ is too small, the estimation of $e(h)$ will be poor and less reliable. On the other hand, if γ is too large, the model will have very few data points to learn from, which can lead to overfitting and poor generalization.

Typically, a common choice for γ is around 0.1, which allows for a reasonable balance between having enough data for estimation and maintaining a sufficient number of samples for training the model effectively. In practice, the direct estimation of $e(h)$ can be noisy. To address this issue, a special type of validation known as k-fold cross-validation is commonly used. In k-fold cross-validation, the hypothesis class $\mathscr{H}$ takes the dataset S and divides it into $S_1, S_2, \ldots, S_k$. For each $i = 1, 2, \ldots, k$, a hypothesis h_i is obtained by minimizing the empirical error on $\bigcup_{j\neq i} S_j$. The error is then estimated by calculating the empirical error $\widehat{e}_{S_i}(h(i))$, and the values are averaged:

$$\widehat{e}(h(i)) = \frac{1}{k} \sum_{i=1}^{k} \widehat{e}_{S_i}(h(i)). \tag{2.36}$$

Figure 11 illustrates the process starting with the division of the dataset into S_{train} and S_{test}. The idea is to take the training set (S_{train}) and divide it into k subsets[9] (in the case of Figure 11, $k = 5$). Once these divisions are made, the performance of the trained model is evaluated using the distribution shown in the figure. Specifically, one of the subsets is selected as the test set (red color), while the remaining $k-1$ subsets are used for training (turquoise color). In the next iteration of the algorithm, the test subset is changed, and training is performed using

[7] Chernoff bounds(Aditive Form): $X_j \in \{0,1\}$, $\mathbf{P}[X_j = 1] = p \Longrightarrow \frac{(X_j - p)}{n} \in \left\{-\frac{p}{n}, \frac{1-p}{n}\right\} \rightarrow b_j - a_j = \frac{1}{n}$

[8] $\mathbf{P}\left[\frac{1}{n}\sum_{j=1}^{n} X_j - p \geq \varepsilon\right] \leq \underbrace{e^{-2\varepsilon^2 n}}_{\delta} \Longrightarrow n = \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}$

[9] Adapted from: <https://scikit-learn.org/stable/modules/cross_validation.html#k-fold>

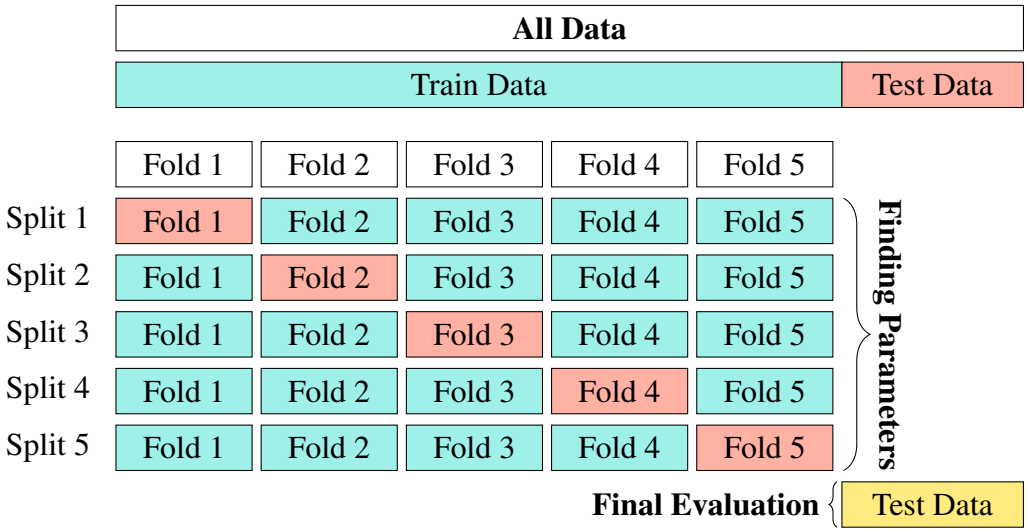


Figure 11 – Illustration of a 5-fold cross-validation process, depicting the splitting of data into training and test sets, the discovered parameters, and the final evaluation.

the remaining subsets. This process continues until each subset has been used as the test subset at least once. To determine the best model, the criterion will be the error it has with the original test dataset S_{test}. The model that has the lowest error will be the one that, through a certain combination of parameters, minimizes the empirical error.

Implementing a multiple cross-validation technique has certain implications that need to be considered. For example, if the hypothesis class $\mathscr{H}$ is a Random Forest algorithm with many parameters, this procedure becomes computationally complex and costly. Although it is widely used in practice, it lacks technical support because it is still an open problem in the research community. Therefore, it holds significant importance for further investigation and advancements.

In the realm of machine learning, "ill-posed" problems represent a unique challenge. Ill-posed problems are defined as those that do not satisfy the three conditions set by mathematician Jacques Hadamard for "well-posed" problems: a solution exists, the solution is unique, and the solution's behavior changes continuously with the initial conditions. In the context of machine learning, these problems arise frequently due to the high-dimensional and often incomplete nature of the data. Examples include overfitting, where an overly complex model is too precisely fitted to the training data and hence fails to generalize well to unseen data, and multicollinearity, where predictor variables in a multiple regression are highly correlated, making it difficult to identify the unique contribution of each. One common solution to tackle such issues involves regularization, where a penalty is imposed on the complexity of the model, effectively reducing its tendency to overfit the data or to be excessively sensitive to individual

predictors. This approach effectively trades off between bias and variance, promoting models that are not just fitting well to the data at hand but also likely to generalize well to new data. Regularization, therefore, is a key strategy in addressing the challenges posed by "ill-posed" problems in machine learning.

So, to see the effects of regularization on the aforementioned ML methods, consider a model with parameters $\mathbf{w}$ and an error function $E(\mathbf{w})$. The regularized error function can be defined as:

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w}), \tag{2.37}$$

where $R(\mathbf{w})$ is the regularization term that penalizes complex models, and $\lambda \geq 0$ is the regularization constant, which is typically tuned during the model regularization process. The idea is to find a balance between fitting the data and model complexity. In terms of the error function, this means reducing variance by introducing some bias.

Before, it was shown that $E(\mathbf{w})$ has a quadratic form, which allows for convex optimization to find the $\mathbf{w}$ that minimize this error function. Now, the focus is on the regularizer term $R(\mathbf{w})$. Let's explore the possible forms that this term can take. We begin by providing an intuition for convex regularization. Consider the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}} \ f(\mathbf{x}) + \lambda g(\mathbf{x}) \\ &\min \ f(\mathbf{x}) \\ \text{subject to} \ \ &g(\mathbf{x}) \leq t_\lambda, \end{aligned} \tag{2.38}$$

where f and g are strictly convex functions. For each $\lambda > 0$, there exists t_λ that makes the two optimization problems shown earlier equivalent.

- $\mathbf{x}^* = \arg\min_{\mathbf{x}} \ [f(\mathbf{x}) + \lambda g(\mathbf{x})]$
- $(\mathbf{x}, \lambda)$ satisfies the Karush Kuhn Tucker (KKT) conditions for the problem:

$$\begin{aligned} &\min \ f(\mathbf{x}) \\ &\text{subject to} \ \ g(\mathbf{x}) \leq g(\mathbf{x}) \end{aligned} \tag{2.39}$$

  1) $g(\mathbf{x}) \leq g(\mathbf{x})$
  2) $\lambda < 0$
  3) $\mathbf{x}^* \arg\min_{\mathbf{x}} \ \mathscr{L}(\mathbf{x}, \lambda) = \arg\min_{\mathbf{x}} \ [f(\mathbf{x}) + \lambda g(\mathbf{x}) - \lambda g(\mathbf{x}^*)]$

4) $\lambda > 0 \Rightarrow g(\mathbf{x}^*) = g(\mathbf{x}^*)$

That means that $\mathbf{x}^*$ is an optimal solution to the second problem.

To illustrate the previous point, we consider a toy problem where the cost function is trivial, such as $f(x) = (x-\alpha)^2 + \lambda(x-\beta)^2$:

$$\begin{aligned} &\min_x \ (x-\alpha)^2 + \lambda\,(x-\beta)^2 \\ &x^* = \frac{\alpha+\lambda\beta}{1+\lambda} \quad g(x^*) = \left(\frac{\alpha-\beta}{1+\alpha}\right)^2 \end{aligned} \tag{2.40}$$

The equivalent constrained problem would be as follows:

$$\begin{aligned} &\min_x \ (x-a)^2 \\ &\text{subject to} \ \ (x-\beta)^2 \leq \left(\frac{\alpha-\beta}{1+\alpha}\right)^2 \end{aligned} \tag{2.41}$$

After considering the implications of convex regularization, the concept of ridge regularization, also known as l_2 regularization, is introduced for the penalized or regularized error function $\tilde{E}(\mathbf{w})$. In order to apply l_2 regularization, the data must be centered and scaled (i.e., normalized[10]), with $w_0 = \bar{y}_i$. Then, the regularized term with l_2 norm can be expressed as follows:

$$E(\mathbf{w}) = \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i\right)^2 + \lambda \left\|\mathbf{w}\right\|_2^2. \tag{2.42}$$

From the previous error function, it can be observed that it penalizes large parameter values. Moreover, with $\lambda \geq 0$, $E(\mathbf{w})$ is a convex function, as demonstrated below by examining the cases when $\lambda = 0$ or $\lambda > 0$:

$$\lambda \geq 0 \begin{cases} \lambda = 0 \ \text{ known case } \ E(\mathbf{w}) = \mathbf{w}^\top \mathbf{H}\mathbf{w} - 2\mathbf{b}^\top \mathbf{w} + c \Longrightarrow \min(\mathbf{w}) : \hat{\mathbf{w}} \text{ satisfies } \mathbf{H}\hat{\mathbf{w}} - \mathbf{b} = 0 \\ \lambda > 0 \ \ \tilde{E}(\mathbf{w}) = \mathbf{w}^\top (\mathbf{H} + \lambda\mathbf{I})\mathbf{w} - 2\mathbf{b}^\top \mathbf{w} + c \Longrightarrow \min(\mathbf{w}) : \tilde{\mathbf{w}} \text{ satisfies } (\mathbf{H} + \lambda\mathbf{I})\tilde{\mathbf{w}} - \mathbf{b} = 0, \end{cases} \tag{2.43}$$

where $\mathbf{I}$ is the identity matrix of the same dimensions as $\mathbf{H}$. If $\mathbf{H} > 0$ is a positive definite symmetric matrix, a comparison between the values of $\hat{\mathbf{w}}$ and $\tilde{\mathbf{w}}$ can be made based on the orthogonal basis decomposition of these vectors.

[10] Normalization is achieved (for example) via Z-score, using: $Z = \frac{X-\mu}{\sigma}$, where Z represents normalized values, X the original values, μ the mean, and σ the standard deviation of the dataset.

$$\hat{\mathbf{w}} = \sum_j \hat{w}_j \mathbf{u}_j \qquad \tilde{\mathbf{w}} = \sum_j \tilde{w}_j \mathbf{u}_j,$$

where $\mathbf{H}\mathbf{u}_i = \alpha_i \mathbf{u}_i$ (principal components!) Substituting into the equation 2.43:

$$-\mathbf{b} + \sum_j \hat{w}_j \alpha_j \mathbf{u}_j = 0 \qquad -\mathbf{b} + \sum_j \tilde{w}_j \alpha_j \mathbf{u}_j + \lambda \sum_j \tilde{w}_j \mathbf{u}_j = 0 \tag{2.44}$$

by the orthonormality of the $\mathbf{u}_i$:

$$\hat{w}_j \alpha_j = \tilde{w}_j \alpha_j + \lambda \tilde{w}_j \Rightarrow \tilde{w}_j = \hat{w}_j \left(\frac{\alpha_j}{\alpha_j + \lambda} \right)$$

$$\alpha_j \gg \lambda \Rightarrow \tilde{w}_j \approx \hat{w}_j \qquad \alpha_j \ll \lambda \Rightarrow \left|\tilde{w}_j\right| \ll \left|\hat{w}_j\right|$$

Based on the previous derivation, it can be concluded that when penalizing the error function with an l_2 regularization, the smaller eigenvalues of $\mathbf{H}$ are shrunk. This is why these methods are known as shrinkage methods. As seen before, even with this regularization, the function remains convex and satisfies the KKT conditions. Therefore, the optimization problem can be expressed as follows:

$$\begin{aligned} &\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i \right)^2 + \lambda \left\| \mathbf{w} \right\|^2 \\ &\min \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i \right)^2 \\ &\text{subject to } \left\| \mathbf{w} \right\|^2 \leq t \end{aligned} \tag{2.45}$$

Another way to select the regularization term $R(\mathbf{w})$ is by using l_1 norm penalty, also known as LASSO regularization. For this, the same considerations as in l_2 regularization must be taken into account, that is, centered and scaled data with $w_0 = \bar{y}_i$. Hence, the l_1 regularization can be expressed as:

$$\begin{aligned} E(\mathbf{w}) &= \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i \right)^2 + \lambda \left\| \mathbf{w} \right\|_1 \\ &= \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}_i - y_i \right)^2 + \lambda \sum_j \left| w_j \right| \end{aligned} \tag{2.46}$$

From the form of l_1 regularization, it can be observed that it also penalizes large parameters, making it another method of shrinkage. For $\lambda \geq 0$, $E(\mathbf{w})$ is a convex function. However, due to the l_1 regularization term being the sum of absolute parameter values, $E(\mathbf{w})$

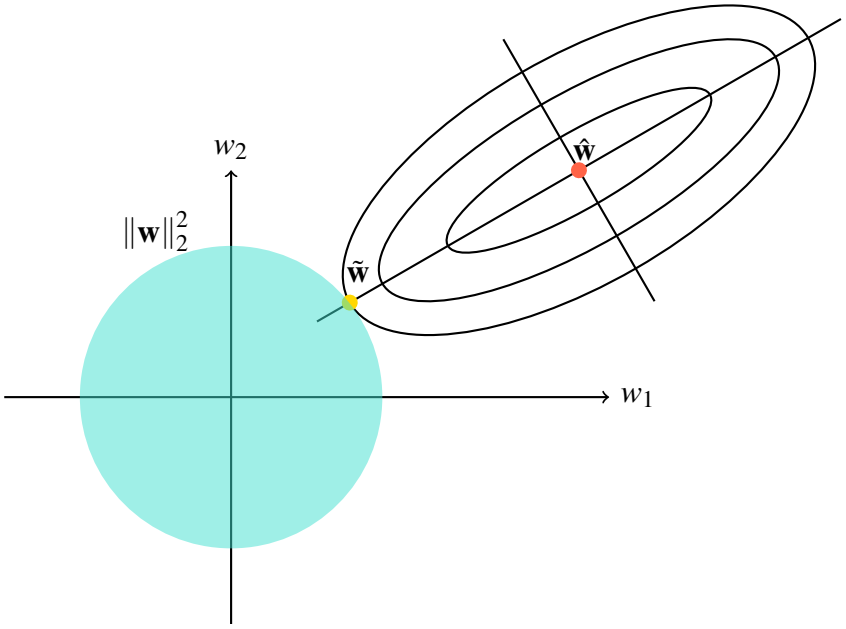


Figure 12 – Estimation picture for the ridge regression. Contours of the error and constraint functions are depicted, with a solid turquoise area representing the constraint region $w_1^2 + w_2^2 \leq t^2$.

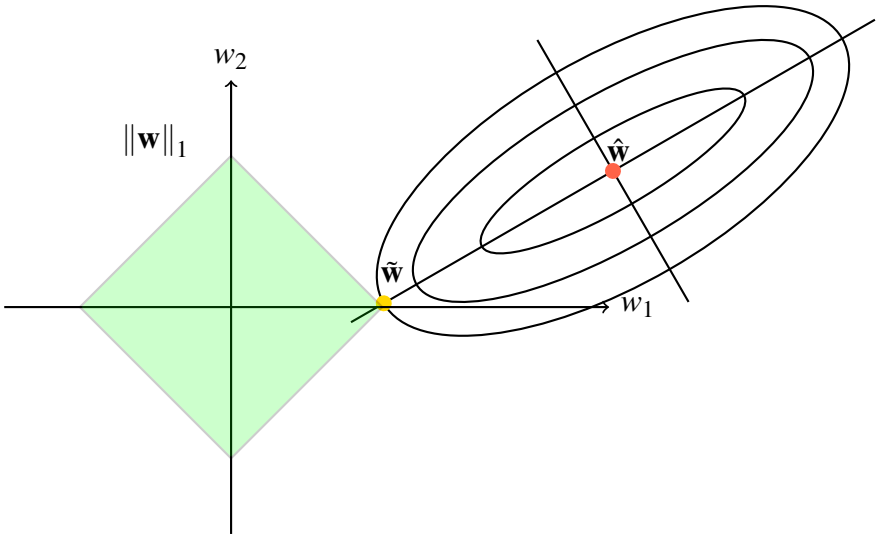


Figure 13 – Estimation picture for the LASSO regression. Contours of the error and constraint functions are depicted, with a solid turquoise area representing the constraint region $|w_1| + |w_2| \leq t$.

becomes a non-differentiable function. This makes l_1 regularization particularly suitable for models with sparse coefficients, where many coefficients are expected to be exactly zero.

Figures 12 and 13 provide a geometric representation of the norms for a two-dimensional model with $\mathbf{w} \in \mathbb{R}^2$. It is evident that the effect of regularization is to restrict the solutions of the model to a circular region in the case of l_2 norm and a diamond-shaped region in the case of l_1 norm. This is demonstrated by the fact that, while the minimum $\hat{\mathbf{w}}$ for the non-regularized problem differs from the regularized problem, the minimum $\tilde{\mathbf{w}}$ is located on the

perimeter of the shaded regions.

To conclude this section, it is worth noting that it is possible to simultaneously perform regularization and model selection, as seen in the earlier part of this section. In fact, regularization restricts the set of models that can fit the data, and as demonstrated, a hyperparameter λ is introduced during regularization, which can be selected through techniques like cross-validation.

2.7 Principal Component Analysis

So far, the emphasis has been on machine learning models that allow for the prediction of labels $\hat{y}_i$ based on a dataset $\mathbf{x}$ in general, where the dataset $\{x_i, y_i\}_{i=1}^{n}$ has a tabular structure. It can be stated that $\mathbf{x} \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, meaning that both the data and labels belong to a Euclidean representation space. The well-known curse of dimensionality was also explained, which in practical terms can be expressed as having more columns than rows in the dataset. This indicates that the nature of the data has high dimensionality compared to the available data points. While all of the above is inherent to the data characteristics, there are some ways to address certain challenges that arise from it. In this section, we will explore a widely used technique known as Principal Component Analysis (PCA) and discuss its advantages and functionalities.

There are two widely accepted definitions of Principal Component Analysis (PCA) that lead to the same algorithm. One definition characterizes PCA as the orthogonal projection of the data onto a lower-dimensional linear space, referred to as the principal subspace. This projection aims to maximize the variance of the projected data (HOTELLING, 1933). Alternatively, PCA can be defined as the linear projection that minimizes the average projection cost, which is measured by the mean squared distance between the data points and their projections (PEARSON, 1901).

To provide a more formal mathematical framework for the definitions of Hotelling and Pearson, Principal Component Analysis (PCA) can be formulated as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} \qquad \begin{cases} \text{Rows of } \mathbf{A} \text{ are data points: } \mathbf{x}_i^\top = [x_{i1}\ x_{i2}\ \ldots x_{in}] \\ \text{Columns of } \mathbf{A} \text{ are descriptors (vectors in } \mathbb{R}^m) \end{cases} \tag{2.47}$$

$$\mathbf{x} \in \mathbb{R}^n \longrightarrow \boxed{\mathbf{T} \in \mathbb{R}^{n \times l}} \longrightarrow \mathbf{y} = \mathbf{T}\mathbf{x} \in \mathbb{R}^l$$

$$l \ll n$$

The previous block diagram illustrates what happens in PCA. We have a dataset $\mathbf{x} \in \mathbb{R}^n$, which exists in a certain dimensional space with dimension n. This dataset undergoes a transformation $\mathbf{T}$, which takes the form $\mathbf{y} = \mathbf{T}\mathbf{x}$, representing a linear transformation where the dimension of the transformed dataset is expected to be lower than the original dimension. The objective then is to preserve the majority of information in the data through this transformation to a lower-dimensional array of size l. However, it is important to note that when reducing the dimensionality of the array, the data no longer retains the same interpretability. Therefore, new descriptors are created, which can be, for example, linear combinations of the descriptors before undergoing the transformation. Now, the necessary procedures for performing PCA are presented. The first step is to center the data:

$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{m}\sum_{j=1}^{m} \mathbf{x}_j,$$

The covariance matrix:

$$\frac{1}{m}\mathbf{A}^\top\mathbf{A} = \begin{bmatrix} \frac{1}{m}\sum_{j=1}^m x_{j1}^2 & \frac{1}{m}\sum_{j=1}^m x_{j1}x_{j2} & \cdots & \frac{1}{m}\sum_{j=1}^m x_{j1}x_{jn} \\ \frac{1}{m}\sum_{j=1}^m x_{j2}x_{j1} & \frac{1}{m}\sum_{j=1}^m x_{j2}^2 & \cdots & \frac{1}{m}\sum_{j=1}^m x_{j2}x_{jn} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{m}\sum_{j=1}^m x_{jn}x_{j1} & \frac{1}{m}\sum_{j=1}^m x_{jn}x_{j2} & \cdots & \frac{1}{m}\sum_{j=1}^m x_{jn}^2 \end{bmatrix} \tag{2.48}$$

Now, if we decompose the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ into its singular values, we have:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$$

$$= \underbrace{\begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r\end{bmatrix}}_{\text{column span basis}} \underbrace{\begin{bmatrix}\sigma_1 & 0 & \dots & 0\\ 0 & \sigma_2 & \dots & \vdots\\ \vdots & 0 & \ddots & \\ 0 & \dots & & \sigma_r\end{bmatrix}}_{\text{singular values}} \left.\begin{bmatrix}\mathbf{v}_1^\top\\ \mathbf{v}_2^\top\\ \vdots\\ \mathbf{v}_r^\top\end{bmatrix}\right\} \text{row span basis,} \tag{2.49}$$

where $\sigma_i = \sqrt{\lambda_i}$, with λ_i being the eigenvalues of $\mathbf{A}^\top\mathbf{A}$. Furthermore, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. Also $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v_r}$ are the eigenvector of $\mathbf{A}^\top\mathbf{A}$ check whats happen with only one data point (Figure 14):

$$\mathbf{x}_i = \underbrace{\left(\mathbf{x}_i^\top \mathbf{v}_1\right)}_{c_1}\mathbf{v}_1 + \underbrace{\left(\mathbf{x}_i^\top \mathbf{v}_2\right)}_{c_2}\mathbf{v}_2 + \cdots + \underbrace{\left(\mathbf{x}_i^\top \mathbf{v}_r\right)}_{c_r}\mathbf{v}_r \tag{2.50}$$

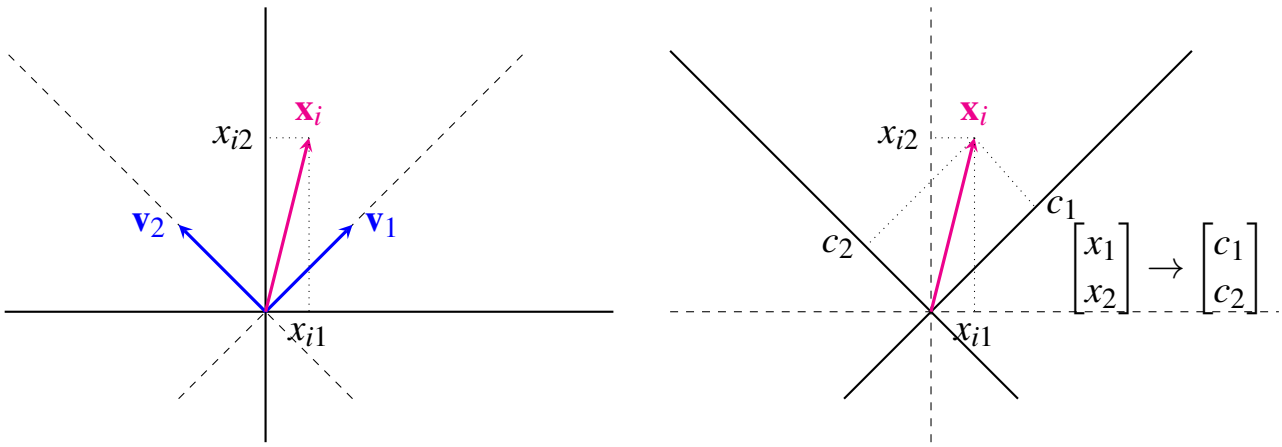


Figure 14 – Projection of a single data point, $\mathbf{x}_i$, onto the basis formed by the eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of Principal Component Analysis, with coefficients c_1 and c_2 representing the coordinates of the point in this new coordinate system.

Taking into account the above, we can generalize the concept for the first principal component as follows:

$$\mathbf{A}\mathbf{v}_1 = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} \quad \mathbf{v}_1 = \begin{bmatrix} \mathbf{x}_1^\top \mathbf{v}_1 \\ \mathbf{x}_2^\top \mathbf{v}_1 \\ \vdots \\ \mathbf{x}_m^\top \mathbf{v}_1 \end{bmatrix} = \sigma_1 \mathbf{u}_1 \rightarrow \text{ Variance } \frac{1}{m} \|\sigma_1 \mathbf{u}_1\|^2 = \frac{\sigma_1^2}{m} \tag{2.51}$$

for two principal component:

$$\mathbf{A}\mathbf{v}_1 + \mathbf{A}\mathbf{v}_2 = \sigma_1 \mathbf{u}_1 + \sigma_2 \mathbf{u}_2 \rightarrow \text{ Variance } \frac{1}{m} \|\sigma_1 \mathbf{u}_1 + \sigma_2 \mathbf{u}_2\|^2 = \frac{\sigma_1^2 + \sigma_2^2}{m}$$

Then for $k \leq n$ principal components:

$$\mathbf{A}\mathbf{v}_1 + \cdots + \mathbf{A}\mathbf{v}_k = \sigma_1 \mathbf{u}_1 + \cdots + \sigma_k \mathbf{u}_k \rightarrow \text{ Variance } \frac{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_k^2}{m} \tag{2.52}$$

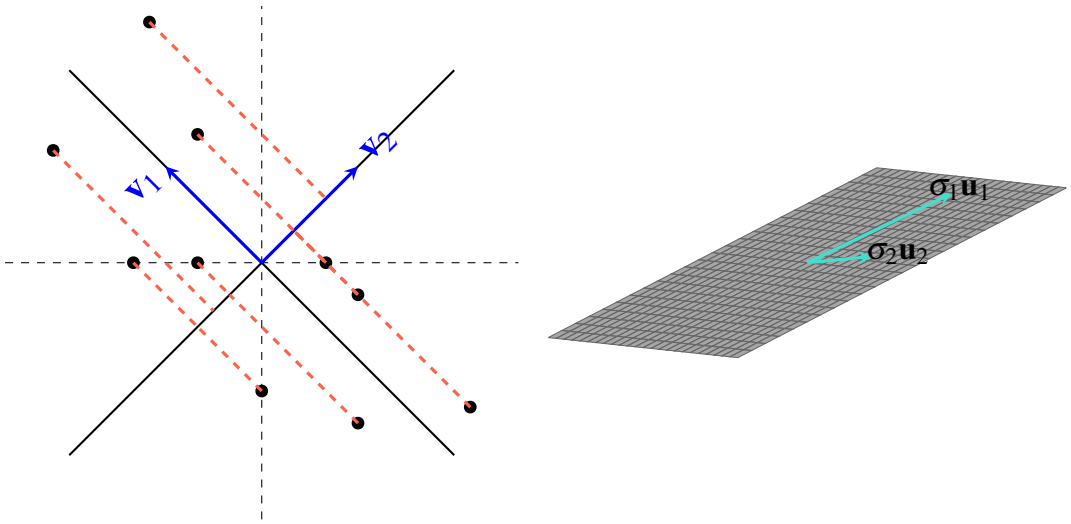


Figure 15 – Visual representation of what happens in PCA. On the left, a scatter plot diagram shows the eigenvectors of the covariance matrix. On the right, the m-dimensional space to which PCA maps the data can be seen.

Figure 15 illustrates the representation of PCA. It shows a dataset, which, for the sake of clarity, is presented in 2 dimensions. As a result, the covariance matrix has only two eigenvalues and eigenvectors, corresponding to two singular values (σ_1, σ_2). When the data is projected onto the basis formed by the eigenvectors, it becomes clear that there is one

dimension where the data has a better representation, as indicated by the higher variance along the eigenvector $\mathbf{v}_1$. In contrast, the projection onto the eigenvector $\mathbf{v}_2$ shows minimal variability. This demonstrates that σ_1 is the singular value that maximizes the variability in the data, leading to the observation that $\sigma_1 \gg \sigma_2$. In general,

$$\sigma_1, \sigma_2, \ldots, \sigma_k \gg \sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_r$$

$$\sigma_1\mathbf{u}_1 + \sigma_2\mathbf{u}_2 + \cdots + \sigma_k\mathbf{u}_k \approx \sigma_1\mathbf{u}_1 + \sigma_2\mathbf{u}_2 + \cdots + \sigma_k\mathbf{u}_k + \sigma_{k+1}\mathbf{u}_{k+1} + \cdots + \sigma_r\mathbf{u}_r$$

then, the new data matrix:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{bmatrix} \underbrace{\begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_k \end{bmatrix}}_{\mathbf{V}_k} = \begin{bmatrix} \mathbf{x}_1^\top\mathbf{v}_1 & \mathbf{x}_1^\top\mathbf{v}_2 & \ldots & \mathbf{x}_1^\top\mathbf{v}_k \\ \mathbf{x}_2^\top\mathbf{v}_1 & \mathbf{x}_2^\top\mathbf{v}_2 & \ldots & \mathbf{x}_2^\top\mathbf{v}_k \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_m^\top\mathbf{v}_1 & \mathbf{x}_m^\top\mathbf{v}_2 & \ldots & \mathbf{x}_m^\top\mathbf{v}_k \end{bmatrix} \tag{2.53}$$

In this way, PCA can be summarized in four steps:

1) Center the data.
2) Find the eigenvalues of the covariance matrix $\mathbf{A}^\top\mathbf{A}$: $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_n^2 > 0$.
3) Choose the value of k that captures the desired percentage of total variance.
4) Compute the projection of the data: $\mathbf{y} = \mathbf{V}_k^\top\mathbf{x}$.

These steps outline the general process of PCA. By following these steps, one can effectively reduce the dimensionality of the data while retaining the most important information captured by the principal components.

So far, we have presented the theoretical foundations underlying widely used ML algorithms. While there are other algorithms that may have similar functionalities, they can have different theoretical underpinnings. However, in practice, these algorithms are often treated as black boxes that take in data and produce predictions. Nonetheless, as we have seen, all techniques have their theoretical foundations, along with their assumptions and conditions.

The question that arises is: why focus on these techniques? The reason is that they form the basis for the algorithms used in the case study we have considered and will continue to study going forward. Understanding the theoretical foundations allows us to grasp the inner workings of these algorithms, interpret their results, and make informed decisions about their applicability and limitations.

3 CASE STUDY

3.0.1 Description of Data and Visualizations.

To construct the datasets, data was sourced from the PubChem database, and molecular information was extracted from JSON files. After filtering, properties of the molecules, such as surface tension, molecular mass, and density, were obtained. To supplement the PubChem data, information on important properties of organic compounds was sourced from Carl Yaws' book (YAWS, 2008). This additional information (Table 1) was then incorporated into the database, filling in the missing quantities.

To extract the data from Yaws' text, Tabula was used to convert the PDF into a CSV file. This allowed for data manipulation using Python libraries such as Pandas and Numpy, which facilitate working with tabular data. Four datasets were obtained, as shown in Table 2. To ensure data consistency, certain conditions were imposed during the data cross-referencing process, such as having a common molecular identifier (the CAS number) in all tables and discarding non-standard information. Additionally, only surface tensions calculated at a temperature of 298.15K were included.

To analyze and extract information from the molecular formula, two libraries, molmass and chemparse, were utilized. This allowed for the incorporation of information on atomic composition and molecular mass into the datasets.

Table 1 – Amount of molecules present in the reference (YAWS, 2008)

Chapter/Feature	No. of Molecules
Enthalpy of Vaporization at Boiling	21337
Density Solid	13781
Critical Properties	9927
Density Liquid	9766
Surface Tension	9766
Thermal Expansion Coefficient	9766
Enthalpy of Vaporization	8646
Enthalpy of Fusion	5846
Dipole Moment	1454
VDW Area and Volume	1411
Radius Gyration	1103
Isothermal Compressibility	650

Hence, our next step is to analyze the provided data from Table 2. Upon inspection of the last row and the distribution of the table, it becomes apparent that the four datasets are not

Table 2 – Datasets generated from the data crossing

Measurement	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Surface Tension	✓	✓	✓	✓
Density Liquid	✓	✓	✓	✓
Enthalpy of Vaporization at Boiling	✓	✓	✓	✓
Enthalpy of Vaporization	✓	✓	✓	✓
Enthalpy of Fusion	✓	✓	✓	✗
Thermal Expansion Coefficient	✓	✓	✓	✓
Dipole Moment	✓	✗	✗	✗
Radius Gyration	✓	✗	✗	✗
VDW Area and Volume	✓	✓	✗	✗
Number of molecules	**498**	**820**	**3377**	**4170**

uniform. They differ in terms of the measured properties and the number of molecules included. Therefore, it is natural to investigate which dataset significantly contributes to the prediction of surface tension in hydrocarbons.

Before proceeding with the data cleaning process, we need to implement a routine that loads each dataset using a parameter and applies a set of procedures. This routine serves two main purposes: i) identifying the best dataset and ii) finding the best classifier for that particular dataset.

The methodology depicted in Figure 16 outlines the approach taken to address the task of surface tension prediction ($\hat{\gamma}$) using a set of features $x_{feats} \in \mathbb{R}^n$. This procedure is applied to all available databases in a consistent manner. Firstly, data cleaning is performed to handle any *NaN* values present in the datasets, which involves discarding compounds with missing information. Next, data preprocessing is carried out, wherein the nature of the data is examined and potential correlations between features are identified. If correlations exist, techniques like principal component analysis (PCA) are applied to transform the data into a space where these correlations are minimized. Subsequently, the data, represented in the optimal dimensionality and quality, is fed into four machine learning algorithms to generate models capable of predicting surface tension based on the provided features.

Considering the planned methodology, it is crucial to assess the dimensions of each dataset to prevent encountering the "curse of dimensionality" (BELLMAN, 2003). The dimensions of the datasets are as follows: the first dataset consists of 497 rows and 22 columns, the second dataset has 820 rows and 20 columns, the third dataset contains 3377 rows and 20 columns, and the fourth dataset comprises 4170 rows and 21 columns. The goal of the proposed work is to identify the dataset that best represents the data and yields accurate predictions of $\hat{\gamma}$

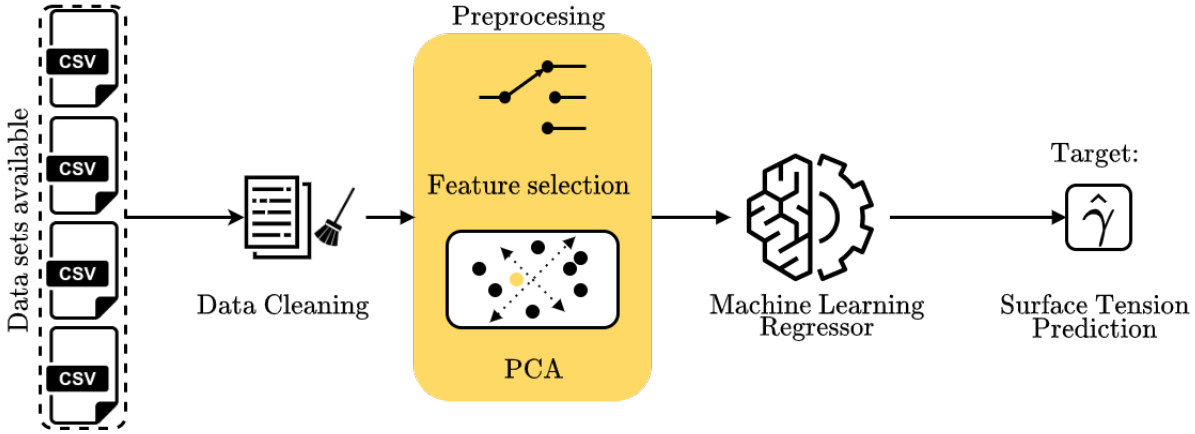


Figure 16 – Block diagram describing the proposed methodology.

using the ML-based regression algorithm that performs optimally under these circumstances. Before implementing any ML algorithm, it is recommended to analyze the correlations among the features (Figure 17). This step is essential as it offers valuable insights into the available data. Although addressing correlations can be challenging in ML algorithms, it can be mitigated by transforming the feature representation space, such as mapping the features to a lower-dimensional space while preserving high variance. In our approach, we propose employing a PCA-based mapping technique to determine its contribution to the regression process. It is important to note that there is a trade-off when applying such techniques, as the interpretability of the original dataset is sacrificed in favor of the mapping process.

After determining the optimal data representation in the methodology outlined in Figure 16, the subsequent task is to choose an appropriate ML algorithm for regression. However, the question that arises is: how can we determine the best regressor? The answer to this query will be discussed below.

3.0.2 Selection of best regressor

To determine the optimal regressor, which refers to the technique that yields the best performance with the most representative dataset, it is necessary to establish a performance measurement criterion for the algorithms. Since the problem at hand involves regression, two metrics are utilized to assess algorithm performance: mean squared error (MSE) and the R^2 score. Therefore, the ideal regressor for the available datasets will exhibit the lowest MSE and the highest R^2 score. To accomplish this, a straightforward routine is proposed, wherein each dataset is loaded individually, and the same procedure is applied to all of them to ensure a fair comparison among the techniques under identical conditions. Algorithm 5 outlines the procedure

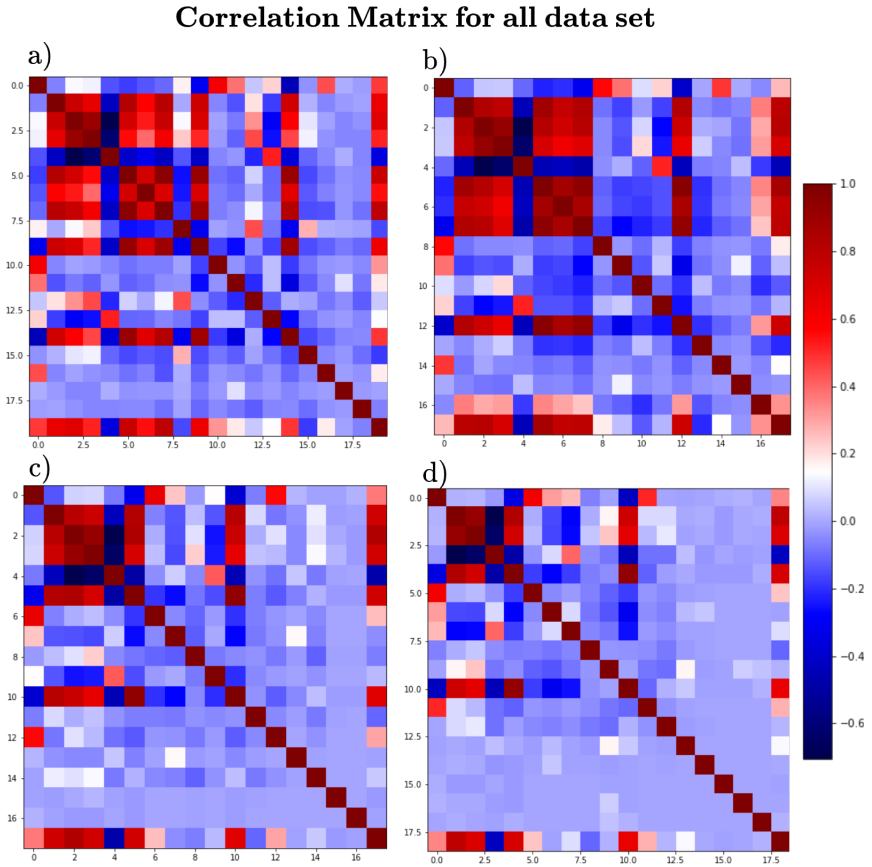


Figure 17 – (a),(b),(c) and (d) represent the correlation matrix of data sets 1, 2, 3, and 4 respectively

for identifying the optimal regressor from the available data. In this case, four algorithms are tested to address the regression problem: a) Linear Regression (LR), b) K-Nearest Neighbor (KNN), c) Random Forest (RF), and d) XGBoost.

Algoritmo 5: Selection of the best algorithm for regression over the available data

Require data_n, $n \in 1,\ldots,4$
All data $\leftarrow$ data_n;
for $i = 1,\ldots,4$ **do**
  All data$[i]$ $\leftarrow$ actual dataset;
  Normalize actual dataset
  Train LR, KNN, RF and XGB over actual dataset
  Store the R^2 and MSE metrics
best regressor $\leftarrow$ max(metrics)

The choice of these algorithms is based on their potential for generalization. Linear regression is used as a baseline for comparing the performance of other algorithms. More complex algorithms may demonstrate improved metrics if linear regression has sufficient generalization capability for the prediction task. The selection of other algorithms is based on their performance

in tabular data, with Random Forest and XGBoost being particularly effective according to the state of the art (CHEN; GUESTRIN, 2016). KNN is included as an intermediate choice to provide additional performance comparisons.

Regarding the data distribution, each available dataset is divided into two subsets: one exclusively for training the Machine Learning algorithms (80%) and the other for testing the models generated by the algorithms (20%). Although not explicitly stated in Algorithm 5, a grid search is conducted for each implementation to explore a set of hyperparameters. Additionally, *k-fold cross-validation* is performed on a portion of the training data, ensuring accurate selection of the best model. In simplified terms, a set of potential regressors is obtained, and the one with the best performance on data not used for training is selected, considering metrics, hyperparameters, and dataset representativeness. Once the best regressor is chosen, the model is tested. This process is repeated for each algorithm to establish the comparisons presented in the Results section.

4 RESULTS

As stated in the methodology, enhancing feature selection can lead to improved performance in the ML algorithms' metrics. Hence, the first subsection of the results will demonstrate the impact of feature mapping to alternative representation spaces, such as PCA, on the algorithm's performance.

4.0.1 PCA Explanation

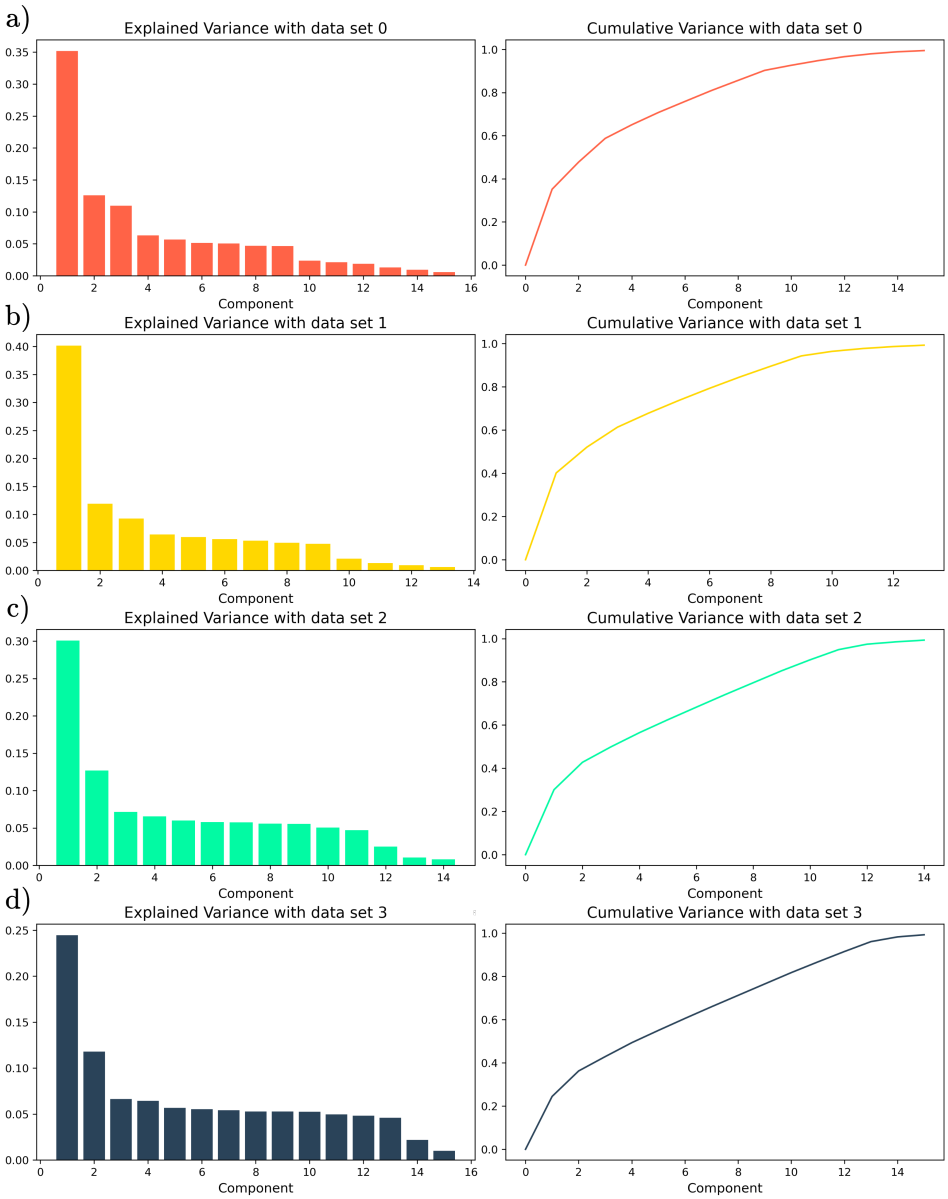


Figure 18 – PCA explanation of cumulative and explained variance per component (a), (b), (c), and (d) for data set 0,1,2 and 3 respectively. This figure provides valuable information on the variance explained by each component in each data set, allowing for a better understanding of the data variability and structure.

The primary objective of PCA is to retain a significant amount of information from the original data while reducing its dimensionality. However, this reduction comes at the cost of losing variability in the dataset. Figure 18 demonstrates that adding components to the dataset increases the variance, indicating that $\sigma^2_{\text{data}} = 1 \iff \mathbf{F}_{\text{PCA}}$ is complete, where $\mathbf{F}_{\text{PCA}}$ refers to the set of features in the PCA representation space. As depicted in Figure 18, the first component of each dataset captures the highest variance. However, in order to preserve a substantial amount of variance, such as close to 0.99, nearly all components are required. This poses a challenge for applying PCA as the reduction in dimensionality comes at the expense of losing interpretability of the original datasets. In this case, it is evident that $\dim(\mathbf{A}_{\text{or}}) \approx \dim(\mathbf{A}_{\text{PCA}})$, where $\mathbf{A}_{\text{or}}$ represents the original matrix with m data points and n dimensions, such that $\mathbf{A}_{\text{or}} = \left[\mathbf{x}_1^\top, \mathbf{x}_2^\top, \ldots, \mathbf{x}m^\top\right]^\top$, where the rows of $\mathbf{A}_{\text{or}}$ are the data $\mathbf{x}_i^\top$ and the columns are the descriptors or vectors in $\mathbb{R}^m$. Since the dimensionality remains nearly the same, training an ML algorithm based on this representation would be redundant. Therefore, this representation is discarded as it loses interpretability while achieving a dimensionality similar to that of the original data, as illustrated in Figure 18. Consequently, the next subsection presents the metrics obtained after training the algorithms.

4.0.2 Machine Learning Performance

The outcomes obtained from the data preprocessing and training of the ML algorithms are presented in Table 3. It is evident from the table that the performance of each algorithm varies across the different datasets, with the third dataset yielding the best results for all implemented techniques. Among the algorithms, XGBoost (XGB) demonstrates the highest performance, indicating superior generalization capabilities on unseen data during training. This is reflected in the R^2 and mean squared error (MSE) metrics, which are shown in the table. The top row represents the MSE values, while the bottom row corresponds to the R^2 scores. The best-performing results are highlighted in bold.

To observe the performance of each technique on unseen data, a random subset of the data is partitioned multiple times. The experiment is repeated 100 times to generate Figure 19, which displays the boxplots for each algorithm and the corresponding metrics. The focus is solely on the third dataset, as it consistently produced the best results with all ML techniques, as previously demonstrated. Notably, XGBoost (XGB) exhibits superior performance compared to the other algorithms in both metrics. It maintains a lower error level, approximately 4.5 (see

Table 3 – ML algorithms performances for all data set available, the results with the best performance are highlighted in bold.

Technique	Dataset 1	Dataset 2	Dataset 3	Dataset 4
LR	642.68	168.26	11.59	19.70
	0.31	-1.20	0.71	0.74
KNN	769.37	14.27	6.62	15.74
	0.17	0.81	0.83	0.79
RF	671.48	10.29	5.46	15.99
	0.28	0.864	0.867	0.795
XGB	707.91	16.05	**4.65**	11.32
	0.24	0.78	**0.887**	0.85

Figure 19b), while achieving the highest prediction score, approximately 0.88 (see Figure 19a). The diamonds depicted in the boxplots represent outliers. The results presented in Table 3 align with the boxplots shown in Figure 19, as the values in the table deviate slightly from the mean in the boxplots but remain close to the expected values.

Now, although the metrics already indicate the ability to generalize from the trained model, further interpretability is desired, especially considering that dimensionality reduction was not performed. In order to gain insights into which features are relevant for predicting surface tension ($\hat{\gamma}$) of hydrocarbons, we can analyze the permutation feature importance. Since XGBoost (XGB) is the best performing classifier on the third dataset, we focus on reporting the importance analysis generated by this algorithm. Figure 20 illustrates that the most important features for XGBoost are vaporization enthalpy, density, molecular weight, and the number of oxygen atoms in the molecule. These features have the highest importance according to the algorithm. However, this result can be debated. While features like density make sense, as surface tension generally decreases with increasing density due to intermolecular forces and interactions, the relevance of hydrogen (H) and carbon (C) composition in the prediction process is not substantial. One possible explanation at the model level is that these two variables are highly correlated, as depicted in Figure 17, which is expected given that hydrocarbons predominantly consist of these two atoms.

It is important to note that not all trained algorithms exhibit the same features with the same level of relevance. In this work, we focus on analyzing and reporting the algorithm that demonstrated the best performance. However, in the supplementary material section, we provide feature importance analyses for the other algorithms as well as the fourth dataset. Figure 21 showcases the most relevant features for algorithms that typically rely on decision trees, particularly Random Forest (RF) and XGBoost. As described in the article, the combination

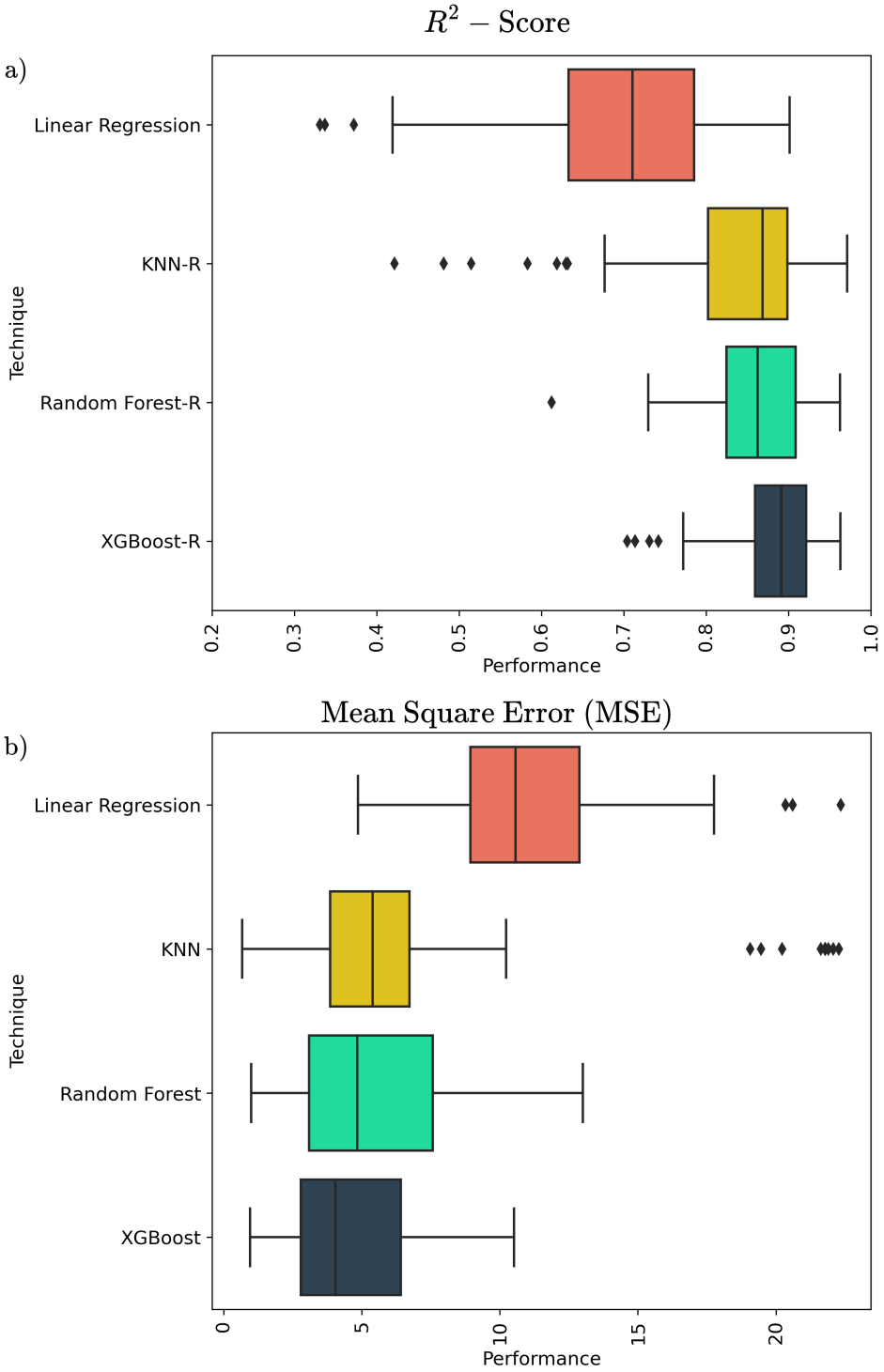


Figure 19 – Box plot depicting the performance of each technique regarding the third dataset, quantifying performance in terms of (a) the R^2 score and (b) mean squared error (MSE).

that yields the best performance is achieved by the Boosting algorithm using the third dataset (Figure 20). It is reasonable to observe the significance of liquid density as a parameter in both regressors. However, Figure 21 highlights that the importance of density varies across the models. Additionally, there are other noteworthy features that are consistently shared among all models,

such as the number of hydrogen atoms, vaporization enthalpy, and molecular weight, as evident in Figure 21.

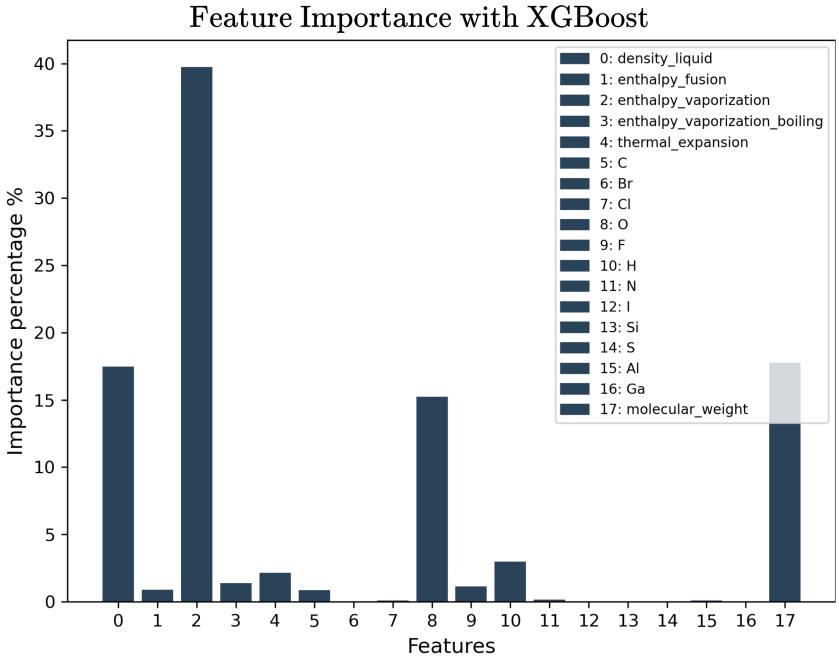


Figure 20 – Feature Importance for the XGBoost algorithm, measured as the percentage of importance in the prediction process of the target variable $\hat{\gamma}$

It is important to highlight that the results of feature importance are only relevant to algorithms that utilize the concept of ensemble models, specifically Random Forest and XGBoost in this study. This analysis would not be meaningful for algorithms like K-Nearest Neighbors (KNN) and traditional linear regression. These models are included in the study primarily as baselines for comparison purposes. KNN, in particular, is not considered state-of-the-art for finite tabular data, which is the case for the consulted databases.

To ensure the reproducibility of our findings, we have made the source code developed for this study available in a dedicated GitHub[1] repository. Researchers and practitioners can access the repository to examine the code and reproduce the results. Additionally, we have also provided the datasets used in this study, which can be accessed for further analysis or to supplement existing research. These resources aim to facilitate transparency, promote collaboration, and encourage further exploration in the field of machine learning for hydrocarbon surface tension prediction.

[1] GitHub Link: <https://github.com/PaulaJimenaFC/Surface_Tension.git>

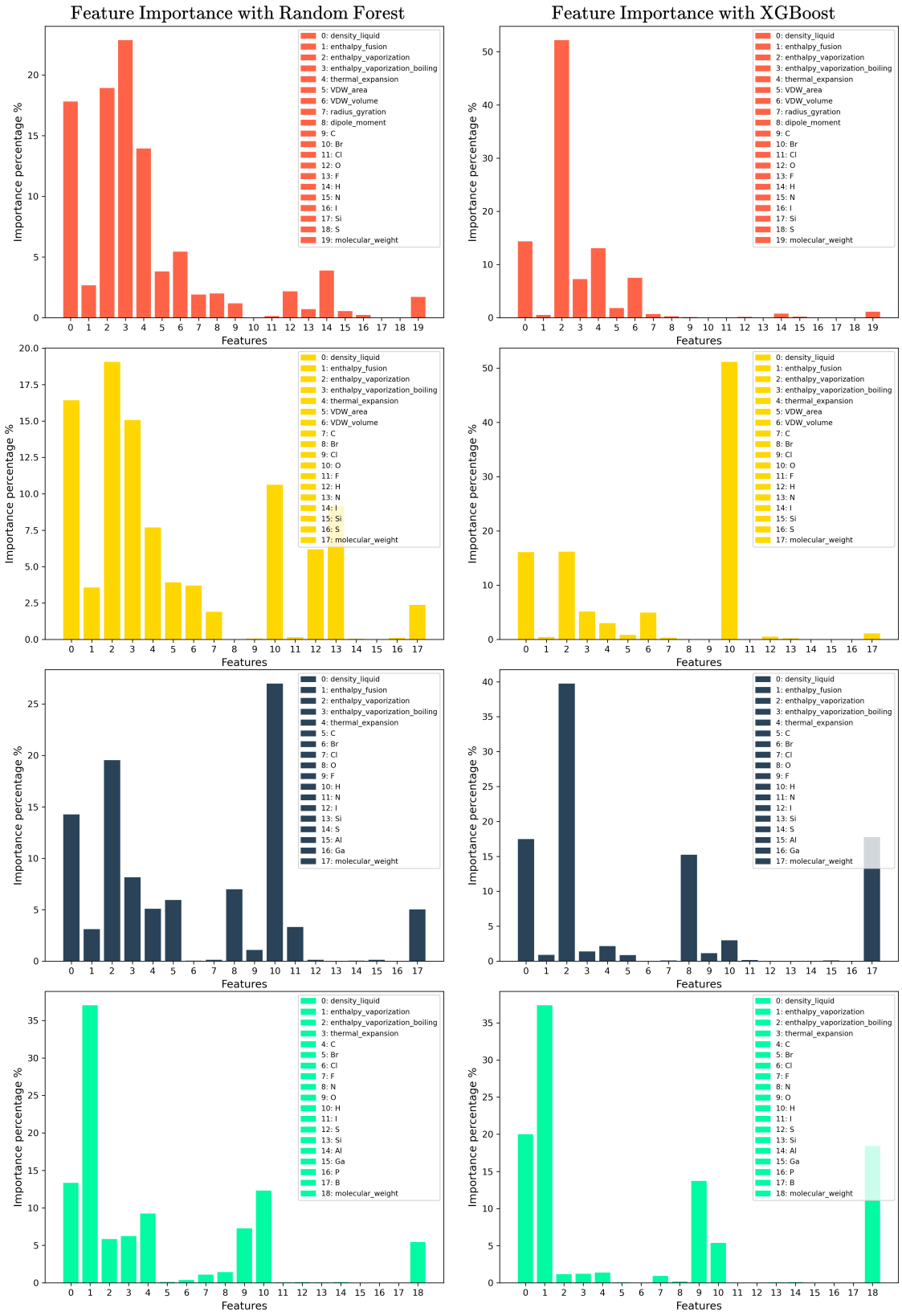


Figure 21 – Importance of the characteristics is highlighted for each of the available data sets. It should be noted that the colors of the bars represent different data sets mentioned in the document. This analysis specifically applies to algorithms based on decision trees, such as RF and XGB. It is evident that while they share some of the most important characteristics, they are not exactly the same.

5 CONCLUTIONS

Understanding the theoretical foundations of each technique in machine learning is a clear advantage when applying these algorithms to any dataset. For example, the concepts of regularization and model validation lead to ideas such as hyperparameter tuning, which is crucial for finding the algorithm with the best performance on the data. Despite the computational cost, conducting hyperparameter search algorithms promotes the selection of the optimal model, based on the strong theoretical foundations discussed. Although machine learning algorithms involve randomness, this theoretical analysis provides insights into good practices to ensure that the model's results align with the developed theory.

Before implementing any machine learning algorithm, it is crucial to carefully examine the nature of the data, including its dimensionality, the amount of available data, the need for data imputation, and the presence of correlations among the features. It is also important to explore how the data behaves in a lower-dimensional space. These considerations are essential because, for instance, to make probabilistic statements about the error, certain guarantees must be met by the data to estimate parameters such as precision (ε) and confidence ($1-\delta$). This implies that the data should be of sufficient quantity to avoid falling into the curse of dimensionality and should also be rich enough for the machine learning algorithms to generalize well to unseen data.

Following a thorough exploration of various machine learning algorithms for predicting hydrocarbon surface tension, it has been observed that XGBoost demonstrates superior performance in terms of predictive accuracy and generalization capabilities compared to other algorithms. The findings highlight the significance of specific molecular properties, including enthalpy of vaporization, molecular weight, density, and the number of oxygen atoms, as key factors in accurately predicting surface tension. These results contribute to a better understanding of the relationship between molecular characteristics and surface tension behavior in hydrocarbons.

While our discoveries shed light on the molecular properties that impact surface tension, there is still room for additional investigation into other molecular attributes and their correlations. This study presents a promising framework for predicting hydrocarbon surface tension, which holds potential applications in diverse fields, including the oil and gas industry and materials science. Further exploration and refinement of these predictive models can contribute to advancements in various practical domains.

REFERENCES

ALMEIDA, B. S.; GAMA, M. M. Telo da. Surface tension of simple mixtures: comparison between theory and experiment. **The Journal of Physical Chemistry**, v. 93, n. 10, p. 4132–4138, 1989.

BAKER, J. A.; HENDERSON, D. Perturbation theory and equation of state for fluids: the square-well potential. **jcp**, v. 47, n. 8, p. 2856–2861, out. 1967.

BARET, J.-C. Surfactants in droplet-based microfluidics. **Lab on a Chip**, Royal Society of Chemistry, v. 12, n. 3, p. 422–433, 2012.

BELLMAN, R. **Adaptive Control Processes**. [*S. l.*]: Princeton University Press, 2015. (Princeton Legacy Library). ISBN 9781400874668.

BELLMAN, R. E. **Dynamic Programming**. USA: Dover Publications, Inc., 2003. ISBN 0486428095.

BISHOP, C. M. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

CALLION, O. M.; TAYLOR, K.; THOMAS, M.; TAYLOR, A. The influence of surface tension on aerosols produced by medical nebulisers. **International journal of pharmaceutics**, Elsevier, v. 129, n. 1-2, p. 123–136, 1996.

CHELAZZI, D.; BORDES, R.; GIORGI, R.; HOLMBERG, K.; BAGLIONI, P. The use of surfactants in the cleaning of works of art. **Current Opinion in Colloid & Interface Science**, Elsevier, v. 45, p. 108–123, 2020.

CHEN, T.; GUESTRIN, C. XGBoost: A scalable tree boosting system. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2016. (KDD '16), p. 785–794. ISBN 978-1-4503-4232-2.

COX, M. F. Surfactants for hard-surface cleaning: Mechanisms of solid soil removal. **Journal of the American Oil Chemists' Society**, Wiley Online Library, v. 63, n. 4, p. 559–565, 1986.

DOBBELAERE, M. R.; UREEL, Y.; VERMEIRE, F. H.; TOMME, L.; STEVENS, C. V.; GEEM, K. M. V. Machine learning for physicochemical property prediction of complex hydrocarbon mixtures. **Industrial & Engineering Chemistry Research**, ACS Publications, v. 61, n. 24, p. 8581–8594, 2022.

FU, D.; LU, J.-F.; LIU, J.-C.; LI, Y.-G. Prediction of surface tension for pure non-polar fluids based on density functional theory. **Chemical Engineering Science**, v. 56, n. 24, p. 6989–6996, 2001.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. [*S. l.*]: Springer, 2009. (Springer series in statistics). ISBN 9780387848846.

HERNÁNDEZ-SORIANO, M. d. C.; PEÑA, A.; MINGORANCE, M. D. Release of metals from metal-amended soil treated with a sulfosuccinamate surfactant: Effects of surfactant concentration, soil/solution ratio, and ph. **Journal of environmental quality**, Wiley Online Library, v. 39, n. 4, p. 1298–1305, 2010.

HERNÁNDEZ-SORIANO, M. del C.; DEGRYSE, F.; SMOLDERS, E. Mechanisms of enhanced mobilisation of trace metals by anionic surfactants in soil. **Environmental Pollution**, Elsevier, v. 159, n. 3, p. 809–816, 2011.

HOTELLING, H. Analysis of a complex of statistical variables into principal components. **Journal of Educational Psychology**, Warwick & York, v. 24, p. 417–441, 1933. ISSN 1939-2176(Electronic),0022-0663(Print).

KATZ, D. L.; SALTMAN, W. Surface tension of hydrocarbons. **Industrial & Engineering Chemistry**, ACS Publications, v. 31, n. 1, p. 91–94, 1939.

LEE, W.; WALKER, L. M.; ANNA, S. L. Competition between viscoelasticity and surfactant dynamics in flow focusing microfluidics. **Macromolecular Materials and Engineering**, Wiley Online Library, v. 296, n. 3-4, p. 203–213, 2011.

MALDONADO-VALDERRAMA, J.; WILDE, P.; MACIERZANKA, A.; MACKIE, A. The role of bile salts in digestion. **Advances in Colloid and Interface Science**, v. 165, n. 1, p. 36–46, 2011. ISSN 0001-8686. Food Colloids 2010 - On the Road from Interfaces to Consumers.

MASSARWEH, O.; ABUSHAIKHA, A. S. The use of surfactants in enhanced oil recovery: a review of recent advances. **Energy Reports**, Elsevier, v. 6, p. 3150–3178, 2020.

MOHARRAM, K.; ABD-ELHADY, M.; KANDIL, H.; EL-SHERIF, H. Influence of cleaning using water and surfactants on the performance of photovoltaic panels. **Energy Conversion and Management**, Elsevier, v. 68, p. 266–272, 2013.

PARIA, S. Surfactant-enhanced remediation of organic contaminated soil and water. **Advances in colloid and interface science**, Elsevier, v. 138, n. 1, p. 24–58, 2008.

PEARSON, K. Liii. on lines and planes of closest fit to systems of points in space. **The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science**, Taylor & Francis, v. 2, n. 11, p. 559–572, 1901.

PERCIVAL, S. L.; MAYER, D.; MALONE, M.; SWANSON, T.; GIBSON, D.; SCHULTZ, G. Surfactants and their role in wound cleansing and biofilm management. **Journal of Wound Care**, MA Healthcare London, v. 26, n. 11, p. 680–690, 2017.

PIRONTI, C.; MOTTA, O.; RICCIARDI, M.; CAMIN, F.; CUCCINIELLO, R.; PROTO, A. Characterization and authentication of commercial cleaning products formulated with biobased surfactants by stable carbon isotope ratio. **Talanta**, Elsevier, v. 219, p. 121256, 2020.

ROKONI, T. S. S. M. R.-M. L. Z. A.; SUN, Y. A machine learning approach for estimating surface tension based on pendant drop images. **Fluid Phase Equilibria**, v. 538, p. 113012, 2021. ISSN 0378-3812.

SCHAPIRE, R. E. The strength of weak learnability. **Machine Learning**, v. 5, p. 197–227, 1990. ISSN 1573-0565.

SEDDON, D.; MüLLER, E. A.; CABRAL, J. T. Machine learning hybrid approach for the prediction of surface tension profiles of hydrocarbon surfactants in aqueous solution. **Journal of Colloid and Interface Science**, v. 625, p. 328–339, 2022. ISSN 0021-9797.

SHENG, J. J. Review of surfactant enhanced oil recovery in carbonate reservoirs. **Advances in Petroleum Exploration and Development**, v. 6, n. 1, p. 1–10, 2013.

SIMPLE Estimation of Bulk Thermodynamic Properties and Surface Tension of Polar Fluids. **Journal of Colloid and Interface Science**, v. 174, n. 1, p. 264–267, 1995. ISSN 0021-9797.

SWARTZ, C. A. The variation in the surface tension of gas-saturated petroleum with pressure of saturation. **Physics**, v. 1, n. 4, p. 245–253, 1931.

TAVAKKOLI, O.; KAMYAB, H.; SHARIATI, M.; MOHAMED, A. M.; JUNIN, R. Effect of nanoparticles on the performance of polymer/surfactant flooding for enhanced oil recovery: A review. **Fuel**, Elsevier, v. 312, p. 122867, 2022.

WU, X.; HUANG, J.; ZHUANG, Y.; LIU, Y.; YANG, J.; OUYANG, H.; HAN, X. Prediction models of saturated vapor pressure, saturated density, surface tension, viscosity and thermal conductivity of electronic fluoride liquids in two-phase liquid immersion cooling systems: A comprehensive review. **Applied Sciences**, v. 13, n. 7, 2023. ISSN 2076-3417.

YAWS, C. **Thermophysical Properties of Chemicals and Hydrocarbons**. [*S. l.*]: Elsevier Science, 2008. ISBN 9780815519904.

ZHOU, Q.; LU, S.; WU, Y.; WANG, J. Property-oriented material design based on a data-driven machine learning technique. **The journal of physical chemistry letters**, ACS Publications, v. 11, n. 10, p. 3920–3927, 2020.