University of Tartu

Faculty of Science and Technology

Institute of Technology

Markus Marandi

Machine Learning Framework for Classification of Potential Hereditary Cancers

Bachelor's thesis (12 ECTS)
Computer Engineering

Supervisor:

MD Villem Pata

Tartu 2024

Abstract

Machine Learning Framework for Classification of Potential Hereditary Cancers

This thesis investigates a machine learning model that classifies potential pathological genetic variants from targeted hereditary data. Due to the vast amounts of data generated in clinical genetics, rapid and precise screening is essential for diagnostics, which can be facilitated by machine learning. The study utilises a dataset from Tartu University Hospital containing genetic variants of 7,498 individuals, including 2,449 investigated due to breast cancer. All genetic variants were reannotated using the Variant Effect Predictor (VEP) database version 111 with allele frequency and pathogenicity scores.

For training the XGBoost-based model, fields such as 'IMPACT' (predicted impact of a genetic variant), 'QUAL' (quality score of the variant call), 'DP' (read depth at the position), 'QD' (quality score normalised by depth), and 'MAX_AF' (maximum allele frequency in populations) were chosen, focusing on those critical for clinical evaluation practice. The study highlights a significant bottleneck in researching rare diseases, characterised by a scarcity of pathogenic genetic variants (signal) compared to common genetic variants (noise). Although the model achieved a moderate overall accuracy of 0.999, it exhibited a high precision of 0.834 but a low sensitivity of 0.401 due to the low signal-to-noise ratio.

The practical output of the model is its utility in automatically filtering out negative cases and highlighting potential positive variants for further analysis. The precision-recall curve provides a more objective depiction of the model's performance than the ROC due to the low signal. While the model significantly reduced the number of rows required for clinical consultation by 99.96%, its ability to detect true positive cases was limited.

The anonymous genetic variant dataset created during this research is an independent study object, enabling the improvement of diagnostics with machine learning models. Future enhancements to this model may include integrating clinical data, additional pathogenicity scores, or linking with other databases.

CERCS: B110 Bioinformatics, B790 Clinical Genetics

Keywords: machine learning, variant annotation, predictive modeling, imbalanced dataset, data reannotation, genomic diagnostics

Lühikokkuvõte

Potentsiaalsete pärilike kasvajate klassifitseerimine masinõppe mudeliga

Masinõppemudel klassifitseerib võimalikke patoloogilisi geenivariante sihtmärgistatult sekveneeritud pärilikkusandmetest. Kliinilises geneetikas tekkivate suurte andmemahtude tõttu on diagnostikaks vajalik suurte andmemahtude kiire ja täpne sõelumine, mida võib lahendada masinõppega. Uurimusobjektiks on Tartu Ülikooli Kliinikumi geenivariantide andmebaas, mis sisaldab 7,498 isiku, sh 2,449 rinnavähi tõttu uuritud isikute geenivariante. Kõik geenivariandid on ajakohastatud alleelide esinemissageduse, patogeensusskooridega, kasutades Variant Effect Predictor-it versioon 111 andmebaasi. XGBoostil põhineva masinõppemudeli treenimiseks valiti väljad 'IMPACT' (geenivariandi eeldatav mõju), 'QUAL' (variandi kvaliteediskoor), 'DP' (lugemissügavus positsioonil), 'QD' (kvaliteediskoor normaliseeritud sügavuse järgi) ja 'MAX_AF' (maksimaalne alleeli sagedus populatsioonis) vastavalt olulistele väljadele kliinilises hindamispraktikas.

Uurimus tõi esile haruldaste haiguste uurimises olulise kitsaskoha, milleks on haruldaste patogeensete geenivariantide (signaal) vähesus võrreldes tavaliste geenivariantidega (müra). Kuigi mudel saavutas keskmise üldtäpsuse 0.999 ning kõrge spetsiifilisuse 0.834, oli sensitiivsus kõigest 0.401, mis tulenes madalast signaali-müra suhtest. Võimalik praktiline väljund mudelile on selle kasutatavus negatiivsete juhtude automaatseks filtreerimiseks ning võimalike positiivsete variantide esiletõstmiseks edasiseks analüüsiks. Ennustamise-tundlikkuse graafik (PR curve) annab mudeli jõudlusest objektiivsema pildi kui Receiver Operating Characteristic-kõver madala signaali tõttu. Ehkki loodud mudel vähendas oluliselt kliiniliseks konsultatsiooniks allesjäänud ridade arvu 99.96%, oli see tegelike positiivsete juhtude tuvastamiseks piiratud võimekusega.

Uurimustöö käigus loodud anonüümsete geenivariantide andmemassiiv on iseseisev uurimusobjekt, mis võimaldab parendada masinõppemudelitega diagnostikat. Käesoleva mudeli täpsust võib tulevikus tõsta kliiniliste andmete kaasamine, rohkemate patogeensusskooride või teiste andmebaasidega sidumine.

CERCS: B110 Bioinformaatika, B790 Kliiniline geneetika

Võtmesõnad: masinõpe, variantide teisendamine, ennustav modelleerimine, varieeruva kvaliteediga andmekogu, andmete teisendamine, kliiniline genoomika

Contents

List of Figures

List of Tables

1 Introduction

Cancer is characterised by the uncontrolled growth and spread of abnormal cells and is a major global health challenge. It is the second leading cause of death worldwide, following cardiovascular diseases, as reported by the World Health Organization (WHO) [1]. Early cancer detection is critical and relies heavily on gene analysis, which elucidates the biochemical activities within tissues and cells. Techniques such as genomic testing are crucial in measuring the activity of genes, thus providing vital data for computational analysis [2].

Hereditary cancer results from genetic mutations passed down from one generation to another within a family. Unlike sporadic cancers, which are caused by genetic changes occurring by chance over a person's lifetime, hereditary cancers are associated with inherited genetic abnormalities that significantly increase an individual's risk. These inherited mutations are present in every cell of the body from birth and can lead to the development of cancer at an earlier age than typically observed in sporadic cases. Genetic sequencing for mutations in high-risk genes like *Breast Cancer gene 1* (BRCA1) and *Breast Cancer gene 2* (BRCA2), which are linked to breast and ovarian cancers in women and prostate cancer in men, can identify individuals carrying these inherited changes, enabling proactive management and tailored surveillance strategies to prevent cancer or detect it at an early, more treatable stage [3].

Next-generation Sequencing (NGS) technology allows for the rapid sequencing of vast amounts of Deoxyribonucleic Acid (DNA) by simultaneously processing millions of small fragments. This method generates extensive data as each fragment is sequenced in parallel, producing detailed information on the genetic composition of an organism. The extensive data output from NGS requires sophisticated computational tools and techniques for effective analysis and interpretation [4]. Sequencing is particularly useful for studying hereditary cancers as it enables the analysis of both inherited and tumor DNA, identifying genetic mutations linked to cancer. This knowledge is crucial for early cancer detection in families with hereditary risks and for tailoring more effective treatment plans [5].

Personalised medicine involves customising treatment based on each patient's unique needs. This approach integrates comprehensive genomic data with phenotype information to tailor prevention, diagnosis, and treatment strategies to individual characteristics. Personalised medicine aims to transform healthcare into a preventive, predictive, and participatory system, enhancing physicians' decision-making processes and treating and preventing diseases [6].

Pathogenicity prediction tools are crucial for interpreting the clinical relevance of genetic variants identified through sequencing. These tools vary widely in their accuracy in categorising genetic variants as benign or likely to cause disease. Research, such as the evaluation performed using data from the Exome Aggregation Consortium (ExAC), shows significant differences in the accuracy of these tools, particularly in detecting benign variants [7]. Tools such as Variant Effect Predictor (VEP) from Ensembl and the ClinGen Pathogenicity Calculator play crucial roles in the genetic analysis process by providing comprehensive annotations of genetic

variants and applying standardised guidelines to evaluate variant pathogenicity [8, 9].

Machine Learning (ML) application in genetics has significantly expanded, enhancing the precision of diagnosing and treating genetic disorders. ML algorithms adeptly analyse extensive genomic datasets, identifying patterns that reveal varying impacts of mutations. By integrating deep learning techniques, these algorithms are increasingly effective at assessing the pathogenicity of genetic variants and facilitating timely and personalised treatment strategies [10]. This integrated approach shows that combining different types of genomic information leads to better results in diagnosing and understanding complex genetic disorders [11].

1.1 Problem Overview

Analysing genetic data manually is time-consuming, which creates significant challenges in modern medicine. Recent advances in genetic technology and lower sequencing costs have dramatically increased the amount of data available for medical research and clinical applications [4]. Technologies like NGS have improved our ability to analyse genetic materials, allowing for quicker and more thorough investigations of the genetic underpinnings of diseases. However, this surge in data also introduces new challenges: while we can now gather vast amounts of genetic information, interpreting this data clinically remains a difficult task for clinicians. As the volume of data grows, there is a crucial need for systems that can effectively sort and prioritise this information, making it easier to identify straightforward cases and those requiring deeper analysis.

Genetic data interpretation, especially in clinical settings, is still developing despite technological advances. ML offers a promising solution to these obstacles. ML algorithms can analyse large, complex genomic datasets to identify patterns that might indicate disease susceptibility or predict disease outcomes. The application of ML in genetics can improve the accuracy of genetic diagnoses and aid in developing personalised treatment strategies, particularly for hereditary cancers. Early detection of monogenic germline cancers is vital as it enables targeted therapeutic strategies and better management of cancer predisposition in individuals.

Currently, the clinical assessment of a patient's genetic data is cumbersome and complicated because of the volume and complexity of the data. Integrating machine learning offers a way to automate and refine the analysis process, leading to more precise and timely diagnoses. However, implementing such technologies comes with its own set of difficulties, including the need for robust models that can accurately interpret the clinical significance of myriad genetic variations. Thus, developing effective ML tools that can provide reliable, actionable insights into genetic data is crucial for advancing personalised medicine and transforming it into a more proactive and predictive healthcare system.

1.2 Objective and Overview of the Thesis

The primary goal of this thesis is to demonstrate the feasibility of using genomic data to develop a ML model for predicting the likelihood of hereditary breast cancer. This proof-of-concept aims to establish a practical approach for analysing genomic data, focusing on enhancing clinical decision-making processes. Another key aspect of this study is to determine the optimal

level of detail for the dataset, which is initially sparse, to improve the model's predictive accuracy and utility. While doing this, the study also aims to prepare the data for more extensive machine learning algorithms and explore the limitations of common methods. This work seeks to see how far conventional tools can go before more complex solutions are needed. It is crucial to understand the capabilities and limits of common, cost-effective methods.

The model will assign a probability score to each data entry, indicating the potential presence of hereditary breast cancer. A decision threshold will be set to accurately identify a high percentage of true positive cases, balancing sensitivity and specificity.

Validating the model is a crucial part of the study, and the study intends to determine its accuracy by assessing the rates of false positives and false negatives at the chosen threshold. This step is important to ensure the model's reliability and usability in clinical settings.

Given that the dataset includes genomes from thousands of patients, the model will specifically focus on genetic markers prevalent in breast cancer cases. This targeted approach aims to improve the model's relevance in identifying hereditary breast cancer. If the current approach proves useful, the developed methodology may benefit other hereditary diseases.

The text in this thesis was proofread using artificial intelligence to streamline the author's original writing.

2 Literature Review

2.1 Machine Learning

ML is a branch of Artificial Intelligence (AI) and computer science that focuses on using data and algorithms to enable AI to imitate how humans learn, gradually improving its accuracy [12].

ML refers broadly to fitting predictive models to data or identifying informative groupings within data. It is particularly useful when datasets are too large or complex for traditional analysis. ML attempts to imitate human pattern recognition using computational methods. In genomics, where the data consists of large-scale genetic information, ML facilitates the analysis of genomic sequences and variations, aiding in predicting disease susceptibility and discovering genetic interactions. [13]

By automating data analysis, ML makes handling vast datasets feasible and ensures the analysis is reproducible and efficient. The increasing importance of ML in biomedicine reflects its capability to transform how biological data is analysed, making it a crucial tool for advancing medical genetics and biotechnology. [13]

2.1.1 Models

ML models can be broadly categorised based on their dependence on human-directed learning processes — this includes whether the learning is driven by rewards, directed by specific feedback, or guided by labelled examples.

1. Supervised Learning: This model utilises labelled datasets to train algorithms. Each element in the dataset is tagged with the correct answer, which allows the algorithm to learn and make predictions by comparing its outputs with the actual answers. This method is highly effective for classification and regression tasks with clear and well-defined relationships between input and output.

2. Unsupervised Learning: In this approach, the algorithm is provided with data that has no labels and is not organised in any predefined way. The model aims to explore the data and find inherent patterns or structures. Unsupervised learning is ideal for discovering hidden data patterns or grouping data into clusters. This method is particularly useful in scenarios where specific outcomes are not known ahead of time.

3. Semi-Supervised Learning: This type combines elements of both supervised and unsupervised learning. It uses a small amount of labelled data alongside a larger volume of unlabeled data. This approach enhances learning accuracy, especially when acquiring

labelled data is expensive or labour-intensive. It's beneficial in situations like medical imaging, where labelling large amounts of data may be impractical.

4. Reinforcement Learning: Here, the learning process is guided by rewards. Algorithms learn to achieve a goal in an uncertain, potentially complex environment. They make decisions by trying out various actions and learning from mistakes based on the feedback received in terms of rewards or penalties. This type is used in various applications such as robotics, gaming, and navigation systems, where the algorithm must make a series of decisions that lead to a specific goal. [14]

Figure 2.1 demonstrates the process of supervised learning where the algorithm iteratively learns from the labeled data. The process begins with the collection of labeled observations (step 1), where each data point is annotated with the correct answer. These labelled data are then split into two sets: a training set and a test set (step 2). The machine learning algorithm uses the training set to learn and adjust its parameters in order to make accurate predictions (step 3). The resulting prediction model is then evaluated using the test set, allowing for the calculation of various statistical measures to assess the model's performance (step 4). This evaluation helps determine how well the model generalises to new, unseen data.

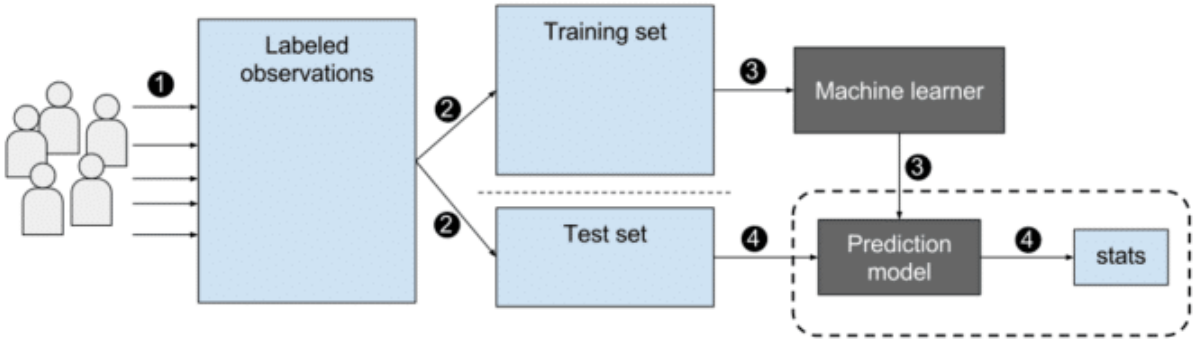

Figure 2.1: With supervised ML, the algorithm learns from labeled data [14]

2.1.1.1 ML Algorithms Used in Supervised Learning

In supervised ML, algorithms are trained on a known dataset and labelled with the correct outcomes to make inferences and predict future events. The learning process involves applying knowledge from this data to new, unseen inputs, with the capability to continually adjust and improve by comparing its outputs to the intended results. Such algorithms excel in analysing large datasets, like those from sequencing technologies, identifying patterns that associate genetic variants with phenotypic outcomes. This is crucial in rare genetic diseases, where the ability to discern complex genomic patterns is instrumental in developing diagnostic and prognostic models, thereby leveraging historical data to anticipate likely future scenarios. [15]

A comparative overview of different supervised ML algorithms and their specific advantages, disadvantages, and applications in the context of genomic data analysis is provided in Table 2.1.

Table 2.1: Comparison of Supervised ML Algorithms for Genomic Data Analysis [16].

Algorithm	**Advantages**	**Disadvantages**	**Applications**
Decision Trees	Easy to interpret, handles feature interactions	Can overfit, struggles with high-dimensional data	Variant classification
Random Forests	Robust to noise, good generalisation	Slow with many trees, opaque results	Genetic association studies
SVM	High accuracy, flexible kernel choice	Complex, slow training speed	Disease prediction models
Naïve Bayes	Simple, works with high dimensions	Assumes feature independence	Variant prioritisation
Logistic Regression	Output as probability, handles nonlinearity	Requires large sample size for stability	Risk prediction
Neural Networks	Can model non-linear relationships, flexible	Black-box, requires a lot of data and computing	Deep phenotype-genotype studies
k-NN	Simple, no need for model training	Slow prediction, sensitive to feature scaling	Similarity searches in genomics

As can be seen from Table 2.1, each supervised ML algorithm has its own unique strengths and weaknesses. Decision Trees, for example, are particularly noted for their interpretability and ability to handle feature interactions. However, they can suffer from overfitting, which is a crucial aspect to manage in order to maintain the validity of models in the context of genomics.

After proper tuning, Decision Trees can accurately classify genetic mutations. As demonstrated in Nascimento et al. (2020), the decision tree model achieved a 92% precision rate in accurately classifying genetic mutations. This high accuracy level confirms Decision Trees' utility in analysing genetic variations relevant to cancer [17].

2.1.1.2 Decision Trees and Neural Networks

Decision trees structure decisions in a tree-like model, where each internal node represents a decision point based on a specific data attribute, branches indicate the outcome of the decision, and each leaf node assigns a classification label. The paths from the root to the leaves define the classification rules. Especially useful in genomics and other data-intensive fields, decision trees provide a clear, straightforward way to interpret complex datasets, breaking down the decision-making process into more straightforward, sequential steps. [18]

Decision trees are valued for their high transparency, as they break down data decision-by-decision at each node based on specific attributes, making the model's decision-making process

straightforward to interpret. While effective for datasets with a limited number of features, decision trees can struggle with complex, high-dimensional data due to their dependence on the quality of features and simple decision rules. In contrast, neural networks are better suited for handling complex, high-dimensional datasets through multiple layers and numerous parameters, allowing them to identify complex patterns in the data. However, this capability comes at the cost of reduced transparency; the depth and interconnectedness of neural network layers often make the underlying logic of their predictions less clear, rendering them a 'black box' when compared to the transparent, rule-based approach of decision trees. This highlights the inherent trade-offs between the simplicity and interpretability of decision trees and the advanced processing power and opacity of neural networks. [19]

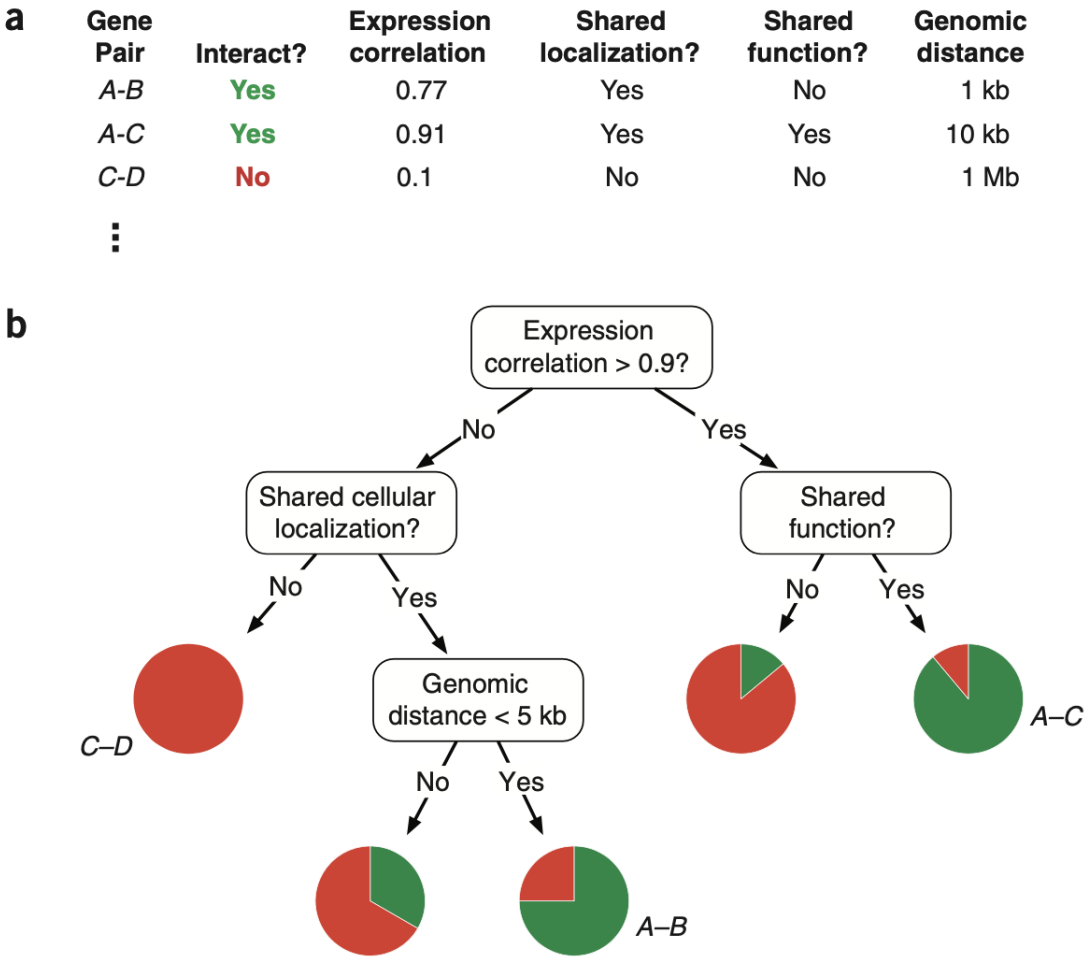

Figure 2.2: Decision tree prediction [20]

Figure 2.2 illustrates a hypothetical example of how a decision tree might predict protein-protein interactions.
(a) Each data item is a gene pair associated with a variety of features. Some features are real-valued numbers (such as the chromosomal distance between the genes or the correlation coefficient of their expression profiles under a set of conditions). Other features are categorical (such as whether the proteins co-localise or are annotated with the same function). Only a few training examples are shown.
(b) A hypothetical decision tree in which each node contains a yes/no question asking about a

single feature of the data items. An example arrives at a leaf according to the answers to the questions.
Pie charts indicate the percentage of interactors (green) and noninteractors (red) from the training examples that reach each leaf. New examples are predicted to interact if they reach a predominately green leaf or to not interact if they reach a predominately red leaf [20].

2.1.2 Advanced ML Techniques

The development of advanced ML techniques has significantly enhanced the ability of models to make accurate predictions across a wide range of complex tasks. These techniques improve traditional methods and introduce novel approaches to model training and prediction. Among these, gradient boosting stands out for its effectiveness in refining predictions through an ensemble of simpler models. EXtreme Gradient Boosting (XGBoost), an optimised version of gradient boosting, further advances this technique by enhancing speed and performance.

2.1.3 Gradient Boosting

Gradient boosting is a method that improves the accuracy of predictions by combining several simple models, usually decision trees. It starts by creating a basic model on the dataset and identifying its prediction mistakes. Then, it adds new models one by one, each designed to correct the mistakes of the one before it. This process continues until adding more models doesn't make much of a difference or a set number of models have been added. In this step-by-step way, each model builds on the previous ones, gradually reducing errors—which is why it's called 'boosting.' By combining these simple models, gradient boosting forms a strong predictor that works well on complex datasets, like those in genomic studies, where usual methods might not be effective. [21]

In gradient boosting, choosing the right loss function is critical - the loss function measures how inaccurate a model's predictions are compared to the actual outcomes. It tells the model how 'wrong' it is, which helps guide it during its training phase to make more accurate predictions. Although the loss function can be chosen based on the specific needs of the task, it plays a key role in how errors are managed and reduced. For example, regression tasks often use the squared-error loss function to minimise the differences between predicted and observed values. This approach helps the model pay more attention to larger errors, significantly enhancing accuracy in predicting numerical outcomes. [22]

2.1.3.1 XGBoost

XGBoost is built upon the foundation of gradient boosting, representing an advanced, optimised implementation known for its high efficiency and scalability across diverse machine-learning tasks. The strength of XGBoost lies in its system design, which allows it to scale across big datasets significantly faster than other implementations. [23]

XGBoost stands out because it effectively addresses overfitting, which happens when a model learns too much detail from the training data, harming its ability to perform well on new data. It does this by simplifying models, which includes limiting the number of decision points (leaves) and reducing the influence of each decision point on the final outcome (leaf weight).

XGBoost also enhances the basic boosting method by improving how the model works with the computer's memory and dealing with missing data or variables that do not have a regular pattern. These improvements help XGBoost work faster and handle larger datasets more efficiently, making it a popular choice in both academic studies and real-world applications. The technique builds models one at a time, each one learning from the errors of the last, and uses a mix of a loss function that points out prediction errors and a regularisation term that keeps the model simple. [23]

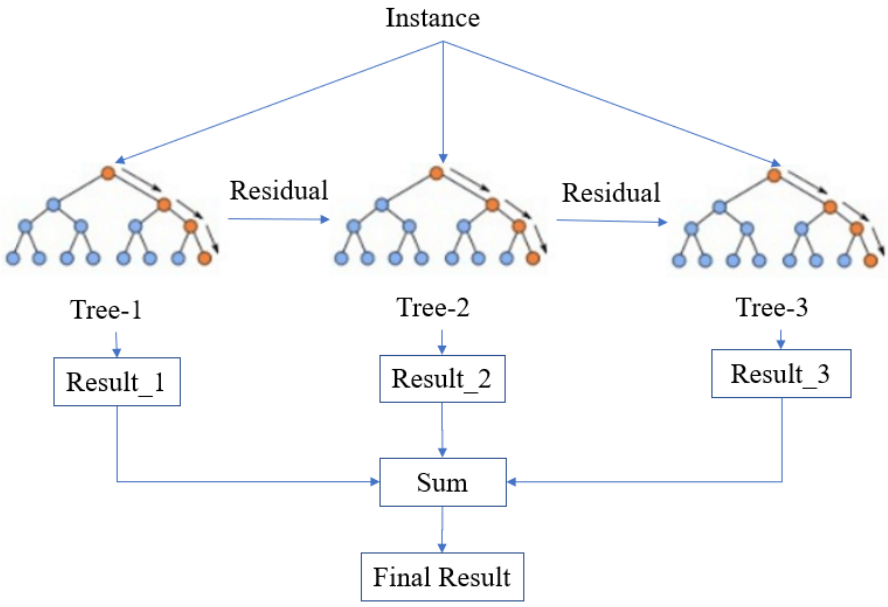

Figure 2.3: The structure of XGBoost illustrating the sequential process where each tree model corrects the residuals of the combined preceding trees, resulting in a refined and improved final prediction [24]

Figure 2.3 illustrates the working mechanism of XGBoost, a powerful supervised learning algorithm used for regression and classification tasks. XGBoost operates through a series of decision trees that iteratively learn from one another. The process starts with Tree-1 making an initial prediction based on the data. The errors from this first prediction, known as residuals, are then passed on to Tree-2. Tree-2 aims to correct these residuals by making its own prediction, thus improving upon the guess made by Tree-1.

This sequential correction continues with each subsequent tree, focusing on the residual errors of the combined predictions of all previous trees. As more trees are added to the model, the predictions become increasingly accurate because each tree incrementally reduces the errors of the model. The final prediction is obtained by summing the predictions of all the trees, leading to a robust model that captures complex patterns in the data. This iterative process of learning and correcting is what makes XGBoost particularly effective in handling large-scale datasets and achieving high predictive performance [24].

2.2 Enhancing Genomic Analyses with ML

ML has proven effective in genomic data analysis, particularly in identifying complex patterns within genome sequences. Initially, the process involves developing specialised algorithms that

utilise supervised learning to understand genomic patterns. These models train on extensive datasets with known outcomes, refining their predictive accuracy over time [25].

After development and training, these ML algorithms are applied to new genomic sequences to make predictions. The effectiveness of the models is evaluated by their accuracy in these predictions. Performance evaluation can occur immediately if a specific test dataset is available; otherwise, additional experimental methods might be needed to validate the predictions, particularly in clinical or research settings [25].

2.2.1 Applications and Challenges

ML algorithms excel in interpreting vast genomic datasets, significantly improving the predictive accuracy of genetic feature identification. Despite their broad application, challenges remain, especially regarding the transparency of deep learning models. These models often act as ”black boxes,” providing results without clear explanations. This can hinder their acceptance by medical professionals who need to trust and understand the tools they use [26].

2.2.2 The Shift Toward Personalised Medicine

The increasing accessibility and affordability of genetic testing propel healthcare towards more personalised treatment approaches based on individual genetic profiles. This evolution highlights the need for ML tools that are accurate, interpretable, and designed to work with diverse and heterogeneous data, such as that found in tabular Variant Call Format (VCF) files. Such developments are crucial as they facilitate the move from traditional symptom-based diagnostics to more predictive and personalised medicine, significantly altering patient care. [6].

Given the transformative potential of ML in genomics, ongoing research must prioritise enhancing the clarity and interpretability of these models. As the field of genetic medicine continues to evolve, it is essential that ML tools not only provide accurate predictions but also offer insights that are comprehensible to healthcare professionals. This approach ensures that personalised treatments can be effectively integrated into clinical practice, improving patient outcomes and improving overall care quality.

2.3 Ethical Considerations and Challenges

The lack of transparency in how ML methods work can make it difficult for clinicians, patients, and policymakers to trust their outcomes. This challenge highlights the importance of clear reporting, as emphasized by Vollmer et al. (2020), to enable independent verification of results and the identification of biases [27]. Protecting data privacy is also paramount due to the use of extensive datasets containing sensitive personal information, which is essential for maintaining trust and adhering to legal requirements [28].

To guide the ethical application of Artificial Intelligence (AI) and ML in healthcare, Vollmer et al. (2020) propose a framework comprising 20 critical questions. These questions cover various stages of ML/AI application, from inception to implementation, stressing the importance of clear documentation and reporting. This approach ensures that research is conducted with a thorough understanding of its impact on patient care and that the results are valuable in practical

clinical environments. Key considerations include evaluating the appropriateness of data for the clinical question, ensuring reproducibility and transparency in the data flow and outcomes, and assessing the potential for models to create or exacerbate disparities in healthcare [27].

ML models can unintentionally exhibit discrimination or bias due to inequalities in the dataset, such as gender, race, or age. This can lead to inequitable healthcare outcomes and worsen pre-existing disparities. To mitigate these biases, it is crucial to train models on diverse and representative datasets. Additionally, the framework includes essential inquiries about data accessibility, the transparency of the modelling approach, and the generalisability of findings beyond the original study settings.

Implementing well-defined guidelines is essential for ensuring the ethical use of AI and ML in healthcare. The 20 critical questions proposed by Vollmer et al. guide ethical practices in AI and ML research, ensuring a clear understanding of their impact on patient care and applicability in real clinical settings [27]. This framework addresses issues such as data validation, methodological transparency, potential biases in outcomes, and the ethical implications of deploying these technologies in real-world scenarios.

While ML and AI have significant potential to improve genomics and personalised medicine, they also present ethical challenges that must be addressed through the implementation of clear guidelines. By confronting these challenges, we can harness the full potential of these advanced tools while upholding ethical principles. Enforcing these guidelines can result in a more ethical use of these technologies, benefiting the public and maintaining high ethical standards.

3 Materials and Methods

This thesis is part of the ”Oligogenic Inheritance in Genetic Diseases” project, funded by the Estonian Research Council and conducted under protocol 362/T-6 of the University of Tartu Research Ethics Committee. It focuses on the reanalysis of genome data initially sequenced for hereditary cancer analysis, involving a review of 7,498 VCF files obtained from Tartu University Hospital. The data analysis was performed in accordance with the data protection plan approved by the Research Ethics Committee and contains no individually identifiable information. Due to restrictions on clinical data handling, the data is not publicly available.

Under the supervision of Principal Investigator Dr. Sander Pajusalu at the University of Tartu, Faculty of Medicine, Institute of Clinical Medicine, this research aims to demonstrate the concept of creating ML models on clinical genomic data and to evaluate the predictive power of these models in the development of genetic diseases.

Our methodology employs Illumina’s TruSight Hereditary Cancer Panel, which sequences the coding regions of approximately 100 genes related to hereditary cancers. This panel has been used from 2015 to 2023 during clinical work, with different versions of Illumina TruSight Cancer and TruSight Hereditary Cancer panels. The VCF files, produced over these years, are collected at the Tartu University Hospital Genetics and Personalised Medicine Clinic to facilitate comprehensive genomic analysis and enhance the understanding of hereditary cancer.

3.1 Data Annotation and Quality Control

3.1.1 Variant Annotation

Our genomic data is stored in the VCF, a standard format for storing DNA sequence variations such as Single Nucleotide Polymorphisms (SNPs), insertions, deletions, and structural variants, enriched with comprehensive annotations. VCF is crucial in genomic research, enabling the sharing and analysis of genetic data. This format efficiently stores large amounts of variant data and metadata, allowing quick retrieval of variant information from specific genome positions. VCF can be used for any diploid genome and represents a wide range of genetic differences compared to a reference sequence. The format is designed for scalability and user customisation, making it suitable for managing genotype information and annotations from thousands of samples. [29]

To enhance our VCF files, we use the Variant Effect Predictor (VEP) for variant annotation. VEP analyses variants to assess their potential impact on gene function and disease manifestation, which is crucial for translating genomic data into clinical insights. This includes evaluating how variants affect different transcripts and protein functions, contributing to the pathogenesis of diseases. By linking genetic variations to specific phenotypic outcomes,

VEP facilitates the transition from general medical practice to personalised medicine, where treatment can be tailored based on individual genetic profiles. [8]

Genome annotation is the process of attaching biological information to sequences. It consists of identifying functional elements within a genome, such as genes, their locations, and their associated regulatory elements. This process is essential because sequencing DNA provides sequences without indicating their function. Over the past decades, genome annotation has advanced significantly—now detailing every nucleotide variation across thousands of genomes, translating raw genomic data into insights that help understand the genetic bases of diseases. This detailed annotation process helps translate raw genomic data into meaningful biological insights. [30]

In this project, we are using GRCh37 as the reference genome. A reference genome is a standardised representation of the human genome sequence that researchers use to compare against DNA sequences generated in their studies. This helps ensure that our analysis is consistent and can be compared reliably with other genetic research. Using GRCh37 allows us to accurately map and analyse the variations in DNA sequences from our study's samples. [31]

3.1.2 Quality control

During the initial processing of the VCF files, we implemented a Quality Control (QC) process to ensure the integrity and reliability of our data. The QC steps aimed to identify and exclude low-quality data that could compromise the accuracy of our findings.

Firstly, we applied a "PASS" filter to retain only those variants that met all the quality criteria established during the variant calling process. This ensures that only high-confidence variants are considered for further analysis. This filter helped us exclude "OffTarget" variants—sequences that do not align with the specific regions of interest in our gene panel. This step is crucial as it focuses the analysis on relevant genetic data and reduces noise from regions outside our scope of study [32].

Furthermore, we excluded "star alleles" from our dataset. Star alleles are variants characterised by complex recombination patterns, making them challenging to classify confidently without detailed analysis. By removing these complex alleles, we ensure our dataset only includes variants with clear and well-defined genetic impacts, enhancing the reliability of our genetic analysis and its application in clinical settings. [33]

3.2 Data Processing

NGS variants are often filtered based on various criteria, including quality scores and read depth [34]. Read depth, defined as the number of times a nucleotide is sequenced, significantly impacts the accuracy of variant calling in sequencing data. Higher read depth enhances the ability to distinguish true genetic variants from sequencing errors, which is crucial in genomic research.

For our analysis, we use Hail, an open-source, Python-based data analysis tool. Hail effectively handles large volumes of VCF files and variant data, streamlining the management of complex datasets. It leverages high read depth to establish the confidence level of each variant reported in the VCF files [35]. In conditions like cancer, where somatic mutations may occur at

low frequencies, adequate read depth, supported by Hail's data processing, ensures accurate detection and reduces the risk of missing significant variants in the analysis [36].

While the VCF is essential in genomic research, it has certain limitations that can complicate the accurate interpretation of data. A notable issue is how genetic variations, particularly structural changes like deletions or duplications, are recorded differently across various databases or tools. For example, the way a deletion is noted can vary between whole genome sequencing data and genotyping arrays. This inconsistency makes it challenging to match and compare genetic variants automatically, limiting effective data integration and analysis. Additionally, VCF primarily records changes from a reference genome, which may exclude other genetic variations present in an individual's DNA. This exclusion might miss important variations that are critical to the analysis. The effectiveness of VCF also relies significantly on the quality and depth of the sequencing data. If the data quality is low or the sequencing depth is insufficient, it could result in important variants being missed or misrepresented. Addressing these issues is crucial for improving the accuracy and usefulness of VCF in both genomic research and clinical applications. [6]

3.3 System Design and Implementation

The detailed system architecture and the specific steps involved in processing, analysing, and modelling the genomic data, is provided in the addendum. This includes comprehensive descriptions of data loading, processing VCF files, generating MatrixTables, data aggregation, ML preparation, model training, and evaluation.

Please refer to the addendum for a thorough explanation of the system architecture and its components.

3.4 System Testing and Validation

System testing and evaluation form crucial aspects of the research methodology applied to ML models. In this thesis, this phase is dedicated to examining the model's capability to identify new and previously unlabelled genetic variants. Utilising a series of established metrics, researchers have the ability to assess the model's reliability and accuracy, confirming its operational effectiveness in practical genomic settings. The metrics evaluated in this study include precision, statistical power, sensitivity, model accuracy, and overall performance. Figure 3.1 illustrates the process of using training, validation, and test data in machine learning workflows.

Training Data

The training data set is used to fit the ML models. It includes a labelled subset of the available data, allowing the model to learn by adjusting weights and biases to minimise prediction errors. The robustness of the training process is crucial for developing a model that performs well not just on the training data but also on new, unseen data. [37]

Test Data

The test dataset serves as the benchmark for evaluating the model. It is employed only after the model has been fully trained (using the training set) and validated (using the validation set). The

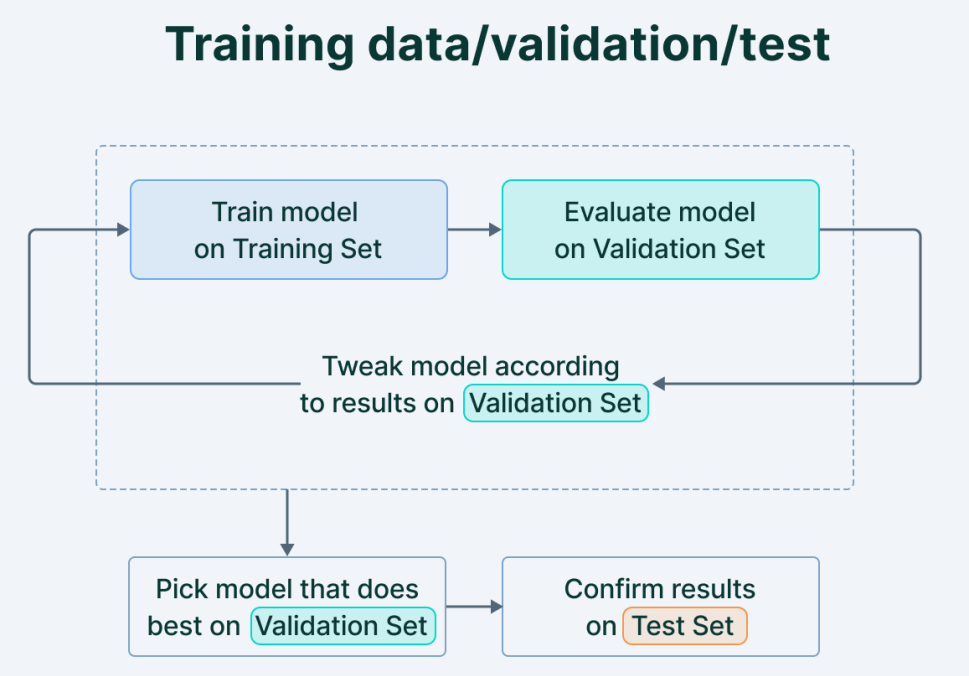

Figure 3.1: Training, validation, test data usage in ML [37]

test data facilitates the assessment of the model's performance under conditions that replicate real-world scenarios [37].

Validation Data
Validation data is used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. This data set helps in preventing models from overfitting and can be used to select the best model architecture or configurations during the training process. [37]

Threshold Adjustment for Classification
During the validation phase, a crucial step involves the adjustment of the decision threshold used for classifying predictions. The decision threshold is a hyperparameter that determines the probability above which a predicted probability is considered a positive result. In this study, the threshold was set at 0.4, rather than the conventional 0.5, to better accommodate the specific characteristics of the dataset and the model's performance objectives. This adjustment allows for a more sensitive detection of positive cases - by lowering the threshold, the model is able to classify more instances as positive, potentially increasing the recall of the model.

3.5 ML Implementation

3.5.1 Feature Selection and Preparation

In preparing the dataset for the application of ML techniques, specific features were selected based on their feasibility for input into the XGBoost model. These features include 'IMPACT', 'QUAL', 'DP', 'QD', and 'MAX_AF', chosen to facilitate the model's ability to identify disease-related mutations. By focusing on these key features, the analysis aimed to enhance the predictive accuracy and interpretability of the results.

The selected features were feasible to include as suitable inputs for the machine learning model:

- **IMPACT**: The impact modifier for the consequence type (can be HIGH, MODERATE, LOW, MODIFIER). This parameter indicates the predicted severity of the mutation's effect on the protein function, with HIGH indicating a likely disruptive effect, and MODIFIER indicating a less certain impact. High-impact variants are more likely to be associated with disease.
- **QUAL**: The quality score from the VCF file (float value). This score indicates the confidence in the variant call, with higher scores representing higher confidence. Reliable variant calls are essential for accurate downstream analysis.
- **DP**: Read depth at this position for this sample (float value). This parameter measures the number of reads that cover a particular genetic position. Higher read depths generally increase confidence in the variant call.
- **QD**: Quality by depth (float value). This value is the variant call quality score normalised by the depth of coverage. It helps to distinguish true variants from sequencing errors, with higher values indicating more reliable calls.
- **MAX_AF**: Maximum observed allele frequency in 1000 Genomes and gnomAD dataset (float value). This provides context on how common the variant is in broader population datasets, which can help assess its potential clinical significance.

3.6 Dataset Characteristics and Classification

3.6.1 Dataset Overview

The VCF files used in this study were collected from 2015 to 2023, each originally annotated at the time of collection with the annotation tools available then (ANNOVAR). Over time, as genomic databases have evolved, these earlier annotations have become outdated. In response, we reannotated all VCF files using VEP, the current standard with a more comprehensive reference genome, ensuring a higher quality of genomic data interpretation. However, this update led to a challenge: some mutation descriptions in the patient phenotype files became inconsistent with the new annotations because the positions of mutations can shift when reference genomes are updated. To address this, we mapped all mutation positions listed in the phenotypes file to both current and potentially older positions using the ClinVar database. This mapping was crucial for accurately identifying mutations within the updated VCF files, ensuring that our analysis remained precise and relevant to current genomic research standards.Table 3.1 presents the distribution of the three most common disease conditions in our dataset, illustrating the prevalence of each condition and highlighting the significant representation of breast and stomach cancer in the collected data.

Table 3.1: Distribution of most common disease conditions in the dataset

Disease	Count	Percentage
Healthy	3746	49.96
Breast Cancer	2449	32.66
Stomach Cancer	630	8.40
Other	673	8.98

3.6.2 Data Classification Process

Following the reannotation of the VCF files, we initiated a targeted search within the data to identify rows that matched the criteria for the positive group. This search was specifically designed to locate and match mutations and gene names as listed in the patient's variants file. This matching process involved searching across all mapped mutation positions to accommodate annotation and genomic reference updates.

When a match was found, that particular row was deemed to contain a disease-causing mutation. Subsequently, these identified rows were consolidated into an aggregated positive group table. Conversely, all rows that did not match were placed into a negative group table.

3.6.2.1 Addressing Unmatched Cases

An additional challenge arose for cases initially classified based on the phenotype file as patients with breast cancer, where the mutation was described, but no matching rows were found in the reannotated VCF files. These patients, despite having a described mutation, had no identifiable matching entries in the current dataset due to discrepancies in mutation positioning or annotation differences. Therefore, the entire VCF dataset for these patients was moved to a validation group. This categorisation was based on the premise that the mutation likely exists but was not detectable with the existing data and annotation capabilities.

By categorising unmatched patients into a validation group and others into a negative group table, our analysis remained comprehensive, addressing both confirmed negatives and potential, yet unverifiable, positives. This approach allowed us to systematically account for the limitations of the genomic data and maintain a thorough analysis despite the challenges.

The scale of data aggregation across the groups is depicted in Table 3.2. This distribution highlights the quantitative disparity between the analysed categories:

Table 3.2: Data Aggregation Across Groups

Group	Number of Rows
Positive Group	155
Negative Group	2,290,746
Validation Group	2,758,383

Among the 355 patients analysed, 155 were successfully matched and had their data aggregated into the positive group table. This indicated a confirmed presence of the disease-related mutation according to our search criteria and current genomic annotations. Conversely, the remaining 200 patients, for whom no matching mutation could be found in the reannotated VCF

files, were allocated to the validation group. Their entire VCF data was preserved for further analysis, highlighting the limitations of current genomic data in identifying disease-specific mutations under the updated reference standards.

The disparity in numbers between the groups presents challenges in genomic data analysis, especially considering that a small fraction of the data indicates disease presence. The validation group, being unlabelled, serves as an important resource for future studies. It allows for using more advanced genomic analysis techniques or updated genomic information to clarify the disease relevance of uncertain cases potentially.

3.6.3 Data Splitting for Model Training and Validation

The dataset was divided into training, testing, and validation sets to evaluate the model's performance accurately and to ensure that it generalises well to new data. The validation group consisted entirely of the previously described validation dataset, which consists of the data that could not be definitively categorised as either positive or negative.

After division, the final distribution of the datasets is summarised in Table 3.3. Following table provides an overview of how the data was allocated across different sets for training, testing, and validation purposes:

Table 3.3: Distribution of Data Across Sets

Dataset	**Number of Rows**	**Percentage of Combined Data**
Combined Data	2,290,901	100%
Training Set	1,603,630	70%
Testing Set	687,271	30%
Validation Set	2,758,383	0%

3.6.4 Evaluation Parameters

Precision

Precision, also known as the positive predictive value, measures the accuracy of the model in identifying true positives. It is calculated using the formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.1}$$

where TP represents true positives, and FP represents false positives. This metric is crucial for determining how well the model can identify correct positive cases without mistakenly labelling negative cases as positive. A high precision score indicates that the model accurately recognises genuine positive variants, which is vital in genomic studies where the accuracy of variant detection can significantly affect biological and clinical decisions. [38]

Statistical Power

Statistical power in this context refers to the model's ability to identify effects when they exist correctly. The formula for statistical power is:

$$\text{Power} = 1 - \beta \tag{3.2}$$

where β is the probability of a Type II error (failing to detect an effect when one exists). High statistical power means there is a greater likelihood that the model will detect true genetic variants when they are present, reducing the chances of missing significant genetic information. [39]

Model Accuracy

Model accuracy represents the overall ability of the model to classify both positive and negative cases correctly. It is calculated with:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.3}$$

where TN represents true negatives. This metric provides a broad view of how well the model performs across all predictions. Nevertheless, employing accuracy as a metric on imbalanced datasets can yield deceptive results. In such instances, a model can attain high accuracy by categorising all data as belonging to the predominant class. [39]

Recall (Sensitivity)

Recall, also known as the true positive rate or sensitivity, measures the model's ability to correctly identify all actual positives. It is calculated using the formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.4}$$

where TP represents true positives, and FN represents false negatives. This metric is particularly crucial in scenarios where the cost of missing a positive (e.g., a disease, a faulty component, or an important signal in data) is high. [39]

Ensuring a high recall rate is crucial in clinical diagnostic tests, as the failure to detect a problem could significantly impact patient treatment. Increasing recall optimisation may lead to a fall in precision since the model might anticipate more positive outcomes, increasing the probability of false positives. Recall is frequently employed alongside precision and the F1 Score to assess the performance of a model thoroughly. Recall is a crucial metric in imbalanced datasets as it specifically measures the detection rate of the minority class. Unlike accuracy, which may be deceptive in imbalanced situations because of numerous true negatives, recall offers a practical evaluation of a model's ability to recognise the less common, often more important, class.

F1 Score

The F1 Score is a harmonic mean of precision and recall, offering a balance between the two by considering both false positives and false negatives. It is particularly useful in situations with imbalanced datasets as it does not overly inflate performance figures due to the majority class. The formula for the F1 score is:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

This metric is crucial when dealing with datasets where false negatives and false positives are more critical to balance. [40]

ROC AUC Score

The Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) measures a model's ability to discriminate between classes across all thresholds, providing an aggregate performance measure. The score ranges from 0 to 1, with 1 indicating a perfect model and 0.5 indicating a no-skill classifier. However, ROC AUC should not be used with heavily imbalanced data. Saito and Rehmsmeier (2015) explain that in such cases, the false positive rate is artificially lowered due to the large number of true negatives, which can give misleading performance evaluations [41]. They recommend using the Precision-Recall Plot as a more informative alternative in these scenarios.

PR AUC Score

The Precision-Recall Area Under the Curve (PR AUC) score evaluates the trade-off between precision and recall across different threshold values. This score is particularly informative in the context of imbalanced datasets, as it focuses on the performance of the minority class. The PR AUC is more sensitive to changes in class distribution and provides a better measure of the positive class's predictive performance. PR AUC plot illustrates the relationship between precision and sensitivity (recall). Unlike the fixed baseline in ROC plots, the PR AUC baseline shifts with class distribution, defined by the ratio of positives (P) to negatives (N) as $y = \frac{P}{P+N}$. For instance, this baseline is 0.5 for balanced class distributions but drops to 0.09 for imbalanced distributions with a P:N ratio of 1:10. Consequently, the AUC score changes with the P:N ratio, providing a more accurate performance measure for imbalanced datasets. Saito and Rehmsmeier (2015) emphasised that precision is an intuitive measure when evaluating binary classifiers on imbalanced datasets, as other measures may fail to capture poor performance in such scenarios. [41].

3.6.4.1 Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a binary classification model. It consists of four outcomes: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The confusion matrix helps to understand the model's accuracy, precision, recall, and overall effectiveness. True positives and true negatives indicate correct classifications, while false positives and false negatives indicate errors. By analysing these

outcomes, various performance metrics such as accuracy, precision, recall, and the F1 score can be derived, providing a comprehensive evaluation of the model's predictive capabilities. Figure 3.2 depicts these concepts through actual and predicted labels, illustrating the generation of the four possible outcomes of the confusion matrix [41].

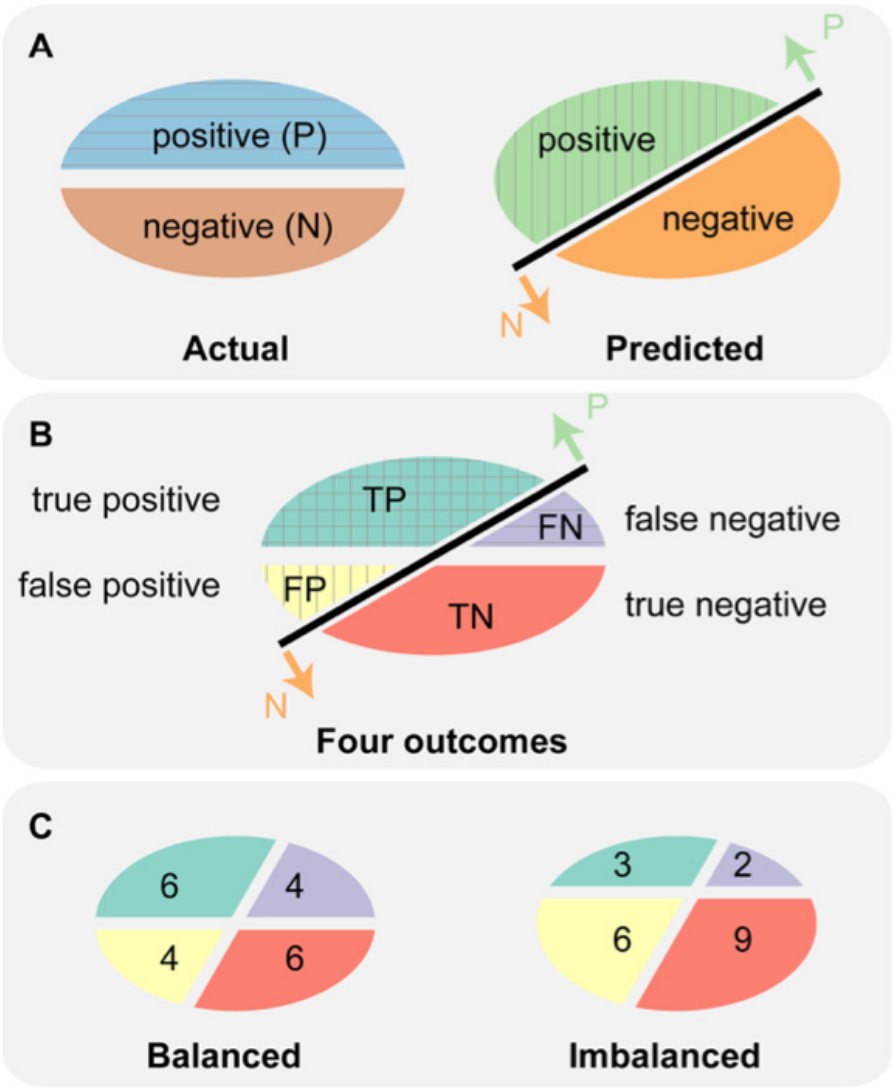

Figure 3.2: Actual and predicted labels generate four outcomes of the confusion matrix [41]

(A) The left oval shows two actual labels: positives (P; blue; top half) and negatives (N; red; bottom half). The right oval shows two predicted labels: "predicted as positive" (light green; top left half) and "predicted as negative" (orange; bottom right half). A black line represents a classifier that separates the data into "predicted as positive" indicated by the upward arrow "P" and "predicted as negative" indicated by the downward arrow "N". (B) Combining two actual and two predicted labels produces four outcomes: True positive (TP; green), False negative (FN; purple), False positive (FP; yellow), and True negative (TN; red). (C) Two ovals show examples of TPs, FPs, TNs, and FNs for balanced (left) and imbalanced (right) data. Both examples use 20 data instances including 10 positives and 10 negatives for the balanced, and 5 positives and 15 negatives for the imbalanced example. [41]

Considerations for Imbalanced Data

When dealing with highly imbalanced datasets, certain performance metrics provide more reliable insights. Metrics such as Precision, Recall, F1 Score, and Precision-Recal (PR) AUC are particularly valuable because they emphasise the performance of the minority class, which is often more critical in practical scenarios. In contrast, metrics like Accuracy and ROC AUC can be misleading. As discussed by Saito and Rehmsmeier (2015), ROC curves can present overly optimistic views of a model's performance on imbalanced datasets because they include the true negative rate, which can dominate the metric when the negative class heavily outweighs the positive class. The study concludes that PRC plots, unlike ROC plots, provide clear visual cues and allow for an accurate and intuitive interpretation of practical classifier performance. [41].

4 Results

4.1 Model Training and Evaluation

Using the XGBoost framework, the model was trained on the training set and evaluated using the testing set. This section will further discuss the specifics of the model training, parameter tuning, and the evaluation metrics employed to gauge the model's performance. Detailed analysis of the training process will highlight the decision-making in model configuration, feature importance, and the impact of different hyperparameters on the model's accuracy and overall effectiveness.

4.1.1 Feature Importance

Feature importance was determined by the XGBoost model based on the gain, which measures the contribution of each feature to the model. Figure 4.1 illustrates the most relevant features identified.

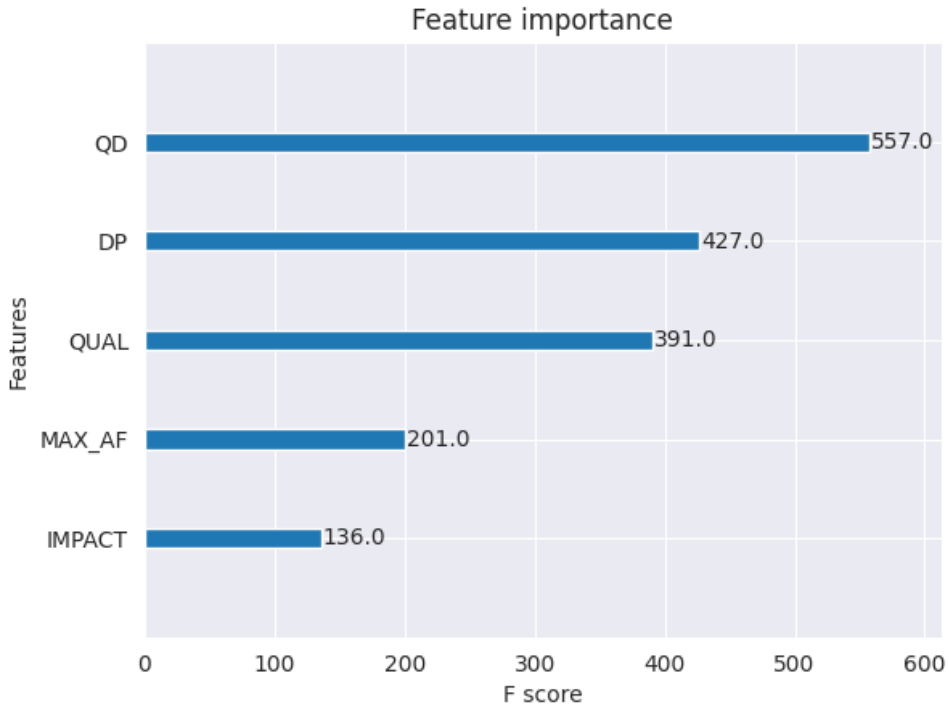

Figure 4.1: Feature importance

The features shown in Figure 4.1 are:

- **QD**: Quality by Depth, which normalises the variant quality score by the depth of coverage.
- **DP**: Read Depth, indicating the number of reads that cover a particular genetic position.
- **QUAL**: Quality score, representing the confidence in the variant call.
- **MAX_AF**: Maximum Allele Frequency observed in population databases such as 1000 Genomes and gnomAD.
- **IMPACT**: Impact modifier for the consequence type, indicating the predicted severity of the mutation's effect on protein function.

These features were selected based on their feasibility in being converted into machine learning-appropriate data.

4.1.2 Confusion Matrix

To further understand the model's performance, a confusion matrix was generated, as shown in Figure 4.2. The confusion matrix provides a detailed breakdown of the true positive, true negative, false positive, and false negative predictions made by the model.

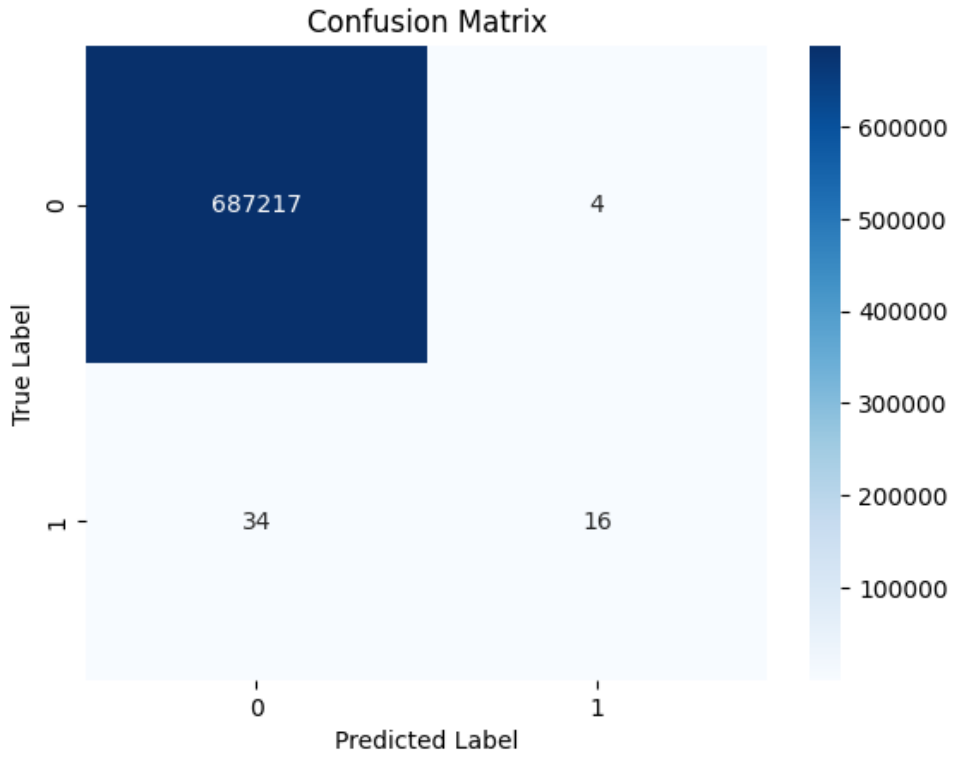

Figure 4.2: Confusion Matrix. The x-axis represents the predicted labels, and the y-axis represents the true labels. The numbers indicate the count of instances for each combination of predicted and true labels.

From the confusion matrix, we observe the following:

- **True Negatives (TN)**: 687,217
- **False Positives (FP)**: 4
- **False Negatives (FN)**: 34
- **True Positives (TP)**: 16

The confusion matrix reveals that while the model has a high number of true negatives, it struggles with identifying true positives. This imbalance is evident from the relatively high number of false negatives (34) compared to true positives (16). This indicates that the model often fails to detect cases of the disease, which is critical in clinical applications where identifying all positive cases is essential.

The high true negative rate suggests that the model is very effective at correctly identifying negative cases. However, the low true positive rate (sensitivity) and high false negative rate highlight a significant limitation in the model's ability to detect positive instances. This is further reflected in the low recall value observed in the evaluation metrics.

4.1.3 Precision-Recall Curve

The PR curve is a crucial evaluation tool for imbalanced datasets, as it provides insights into the trade-offs between precision (the accuracy of the positive predictions) and recall (the ability to find all positive instances). The PR curve for our model is shown in Figure 4.3.

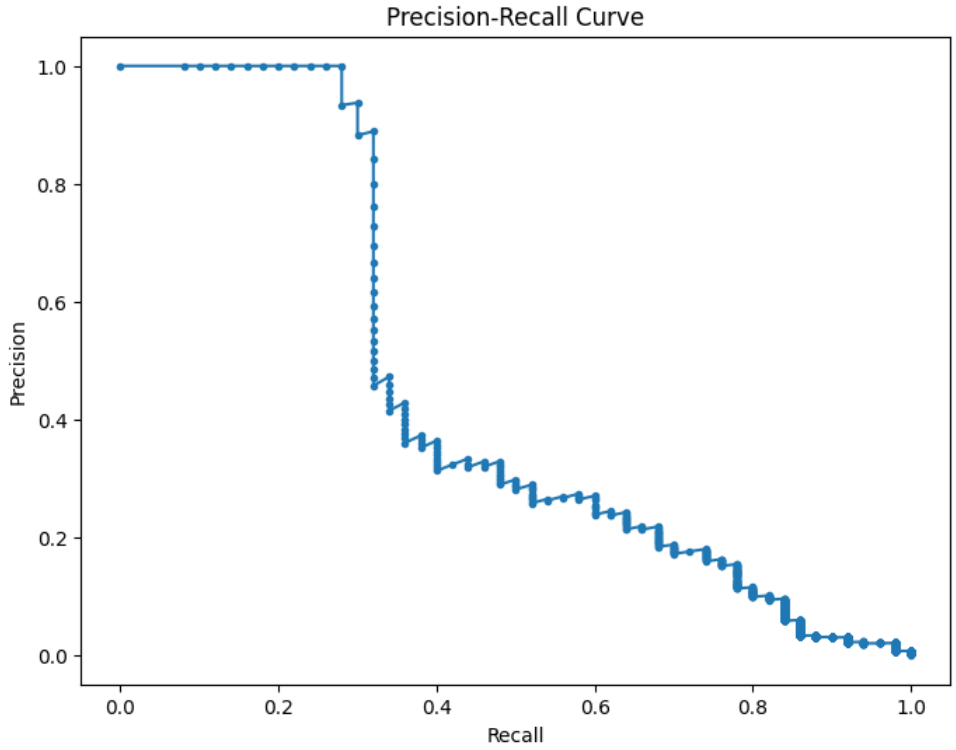

Figure 4.3: Precision-Recall Curve

The PR curve in Figure 4.3 shows how precision and recall vary as the decision threshold is adjusted. A key observation from the PR curve is the model's performance across different threshold values:

- At the leftmost part of the curve, precision is high, but recall is low. This indicates that the model is very conservative in predicting positive cases, leading to fewer false positives but missing many true positives.
- As we move to the right along the curve, recall increases while precision decreases. This suggests that the model is becoming more inclusive in its positive predictions, capturing more true positives but also increasing the number of false positives.
- The sharp decline in precision around a recall value of 0.4 indicates a threshold beyond which the model's precision significantly drops. This highlights the challenge in balancing precision and recall in the presence of imbalanced data.

The PR curve provides valuable insights that complement the confusion matrix analysis. While the confusion matrix offers a snapshot of performance at a specific threshold, the PR curve illustrates how performance varies with different thresholds.

4.1.4 Receiver Operating Characteristic Curve

The ROC curve is a graphical representation that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve for our model is shown in Figure 4.4.

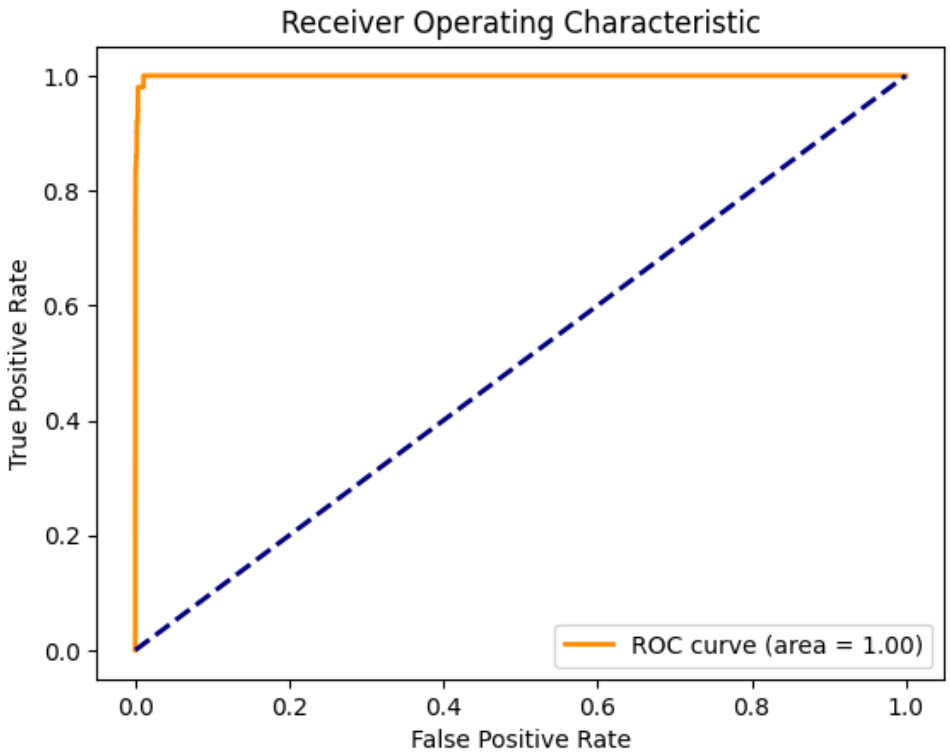

Figure 4.4: Receiver Operating Characteristic Curve (The x-axis represents the false positive rate and the y-axis represents the true positive rate).

Given the imbalanced nature of our dataset, the PR curve is a more informative evaluation metric compared to the ROC curve. It focuses on the performance with respect to the positive

class, which is critical in clinical applications where the detection of disease-related mutations is essential.

The ROC curve shows an area under the curve (AUC) of 1.00, indicating a perfect classifier with no false positives or false negatives.

4.1.5 Model's metrics

The model's performance was assessed using several evaluation metrics as shown in table 4.1.

Table 4.1: Model Evaluation Metrics

Metric	**Value**
Mean ROC AUC	0.9992
Mean Accuracy	0.9999
Mean Precision	0.8342
Mean Recall	0.40091
Mean F1 Score	0.5171
Mean Log Loss	0.0002

Due to the highly imbalanced data, Test Accuracy and ROC AUC are not relevant in our research.

For the validation dataset, the model with a decision threshold of 0.4 was able to correctly identify 7 patients out of 200, and with a precision of 0.38, it potentially identified 1,018 rows. The validation set had 2,758,384 rows, meaning the model significantly reduced the number of rows for clinicians to review. Specifically, the model reduced the number of rows from 2,758,384 to 1,018, which is a reduction of approximately 99.96%.

The decision threshold of 0.4 was chosen based on optimising the balance between precision and recall in our imbalanced dataset. Lowering the threshold from the conventional 0.5 to 0.4 increases the model's ability to capture true positives while maintaining a manageable number of false positives for further clinical review.

$$\text{Percentage Reduction} = \left(1 - \frac{1018}{2758384}\right) \times 100 \approx 99.96\% \tag{4.1}$$

4.2 Summary

4.2.1 Analysis of Results

The ROC curve in Figure 4.4 plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold settings. Key observations from the ROC curve include:

- The ROC curve is very close to the top left corner, indicating that the model has a high true positive rate and a low false positive rate.

- The steep rise to a true positive rate of 1.0 with a very low false positive rate suggests that the model performs well in distinguishing between the positive and negative classes.

While the ROC curve indicates that the model has good discriminative ability between the positive and negative classes, the PR curve and confusion matrix provide more nuanced insights into the model's performance, particularly regarding the detection of positive cases. Given the imbalanced nature of the dataset, the PR curve is more informative in assessing the model's ability to correctly identify positive cases. Future work will focus on balancing these metrics to ensure robust model performance in clinical settings.

4.2.2 XGBoost Decision Tree

To gain a deeper understanding of the decision-making process of the XGBoost model, we visualised one of the trees from the model. The detailed decision tree is provided in Appendix A. By visualising the tree, we can see the hierarchical structure of decisions and how different features contribute at various levels. This visualisation aids in interpreting the model's behaviour and identifying which features are most influential in predicting the outcomes.

4.2.2.1 Analysis of the Decision Tree

The XGBoost decision tree provides a visual representation of how different features and their values contribute to the final prediction. Key observations from the decision tree include:

- **Root Node**: The root node splits on the feature 'IMPACT' with a threshold of 1. This indicates that 'IMPACT' is the most significant feature at the top of the tree.
- **Subsequent Nodes**: The tree further splits on features such as 'QD', 'DP', 'QUAL', and 'MAX_AF'. These features were identified as important in the feature importance analysis and are used to make more refined decisions down the tree.
- **Leaf Nodes**: The leaf nodes contain the final prediction values. These values represent the model's output for the samples that reach that particular leaf.
- **Thresholds and Decisions**: Each node represents a decision point based on a specific threshold value for a feature. For example, one of the decisions is based on whether 'DP' is less than 182.

5 Discussion

This study has identified several critical aspects of genomic data analysis for detecting disease-related mutations, particularly the challenges associated with data reannotation and model performance in an imbalanced dataset. The reannotation of VCF files and the subsequent identification of mutation positions underscored the necessity of maintaining up-to-date genomic annotations to ensure accurate and relevant data interpretation. These efforts were crucial for refining our ML model and improving the detection of clinically significant mutations.

5.1 Dataset Characteristics and Challenges

One of the primary challenges faced during this study was the reannotation of VCF files to ensure higher quality genomic data interpretation. This reannotation introduced inconsistencies between the mutation descriptions in the patient phenotype files and the new annotations. Addressing this required mapping all mutation positions listed in the phenotype files to both current and older positions using the ClinVar database, ensuring that our analysis remained precise and relevant to current genomic research standards.

Another challenge was handling imbalanced data. The dataset contained significantly more negative instances than positive ones, which affected the model's ability to accurately detect true positive cases. Strategies to mitigate this included categorising unmatched patients into a validation group and others into a negative group, ensuring a comprehensive analysis that addressed both confirmed negatives and potential, yet unverifiable, positives.

5.2 Feature Selection and Model Training

For the ML implementation, specific features were identified as most relevant for training the XGBoost model, including IMPACT, QUAL, DP, QD, and MAX_AF. The dataset was divided into training, testing, and validation sets to evaluate the model's performance accurately and ensure generalisation to new data. The validation group consisted of data that could not be definitively categorised.

5.2.1 Model Performance

Using the XGBoost framework, the model was trained on the training set and evaluated using the testing set. Feature importance was determined based on the gain, measuring the

contribution of each feature to the model. The confusion matrix and precision-recall curve were generated to evaluate the model's performance comprehensively.

The confusion matrix revealed a high number of true negatives (687,217) but a low number of true positives (16), indicating a significant number of false negatives (34). This imbalance suggests that the model often fails to detect cases of the disease, which is critical in clinical applications. Improving the model's ability to detect true positives while maintaining a low false positive rate will be crucial. Strategies to address this include further tuning the model, rebalancing the dataset, or exploring alternative algorithms better suited for imbalanced datasets. These strategies go beyond the scope of the current thesis but are essential for future research to enhance the model's clinical utility.

The precision-recall curve provided insights into the trade-offs between precision and recall. Initially, precision is high but recall is low, indicating conservative positive predictions. As recall increases, precision decreases, highlighting the challenge of balancing these metrics in imbalanced data. The sharp decline in precision around a recall value of 0.4 emphasises the need for careful threshold selection to optimise the trade-off between precision and recall.

5.2.2 Model Limitations and Future Directions

The model's performance, while demonstrating high overall accuracy, indicated significant limitations in detecting positive cases, reflected in the low recall and F1 score. Future experiments should focus on converting more genomic data columns to numeric forms to improve model training. Additionally, considering the mutation's effect and rarity, rather than its specific position, may provide better insights into disease-related mutations.

Future studies should include more parameters such as CADD (Combined Annotation Dependent Depletion) scores, which predict the deleteriousness of variants, and PolyPhen (Polymorphism Phenotyping) scores, which assess the potential impact of amino acid substitutions on protein function. Additionally, incorporating parallel annotations from more databases, or even using natural language processing to match phenotype to disease, will enhance the robustness and accuracy of the model. This comprehensive approach will improve the model's predictive power and clinical applicability.

A crucial aspect in model training was the conversion of group labels to numeric values, facilitating model training and improving the model's interpretability. The major strength of this work lies in the data handling, unification, and the dataset itself, which pools hereditary cancers.

5.2.3 Enrichment and Filtering

During this study, the phenotypes file was enriched with mappings to the ClinVar database, enhancing the dataset's quality. Additionally, a PASS filter was applied to filter out off-target rows, which might filter out important data, especially considering mutations mapped to older reference genomes. This highlights the need for careful consideration of filtering criteria to avoid excluding relevant data.

5.2.4 Challenges with Quality Depth (QD)

Quality depth (QD) is crucial because most likely pathogenic variants are only reported if they have sufficient depth of coverage and other quality parameters agree. This causes an issue for the classification of outliers of low quality, which is easy for humans but problematic for ML tuning. Improving how models handle low-quality data points is essential for better performance.

5.2.5 Design Problems and Results

A design problem was identified as the model only detected 7 positive variants with a 0.4 threshold. This highlights the challenge of setting appropriate thresholds in the presence of imbalanced data. Addressing this requires further refinement and possibly the inclusion of more comprehensive data points and advanced analytical techniques. Additionally, a robust model should be agnostic to gender, but our dataset is likely skewed, posing another challenge.

Another design problem is the focus on breast cancer, while the validation set also included other cancer variants. This mixed validation set makes it difficult to trace back to the original mutations, introducing potential inaccuracies. The evaluation was done on real-world model performance based on a pooled untested validation dataset, which may contain more pathogenic variants than initially expected. As a result, we might incorrectly classify some true positives as false positives. At this moment, we cannot verify this due to the inability to trace back to the original VCF files.

These issues underscore the need for a more comprehensive and accurately labelled validation set to better assess model performance and refine our approach.

5.3 Summary of Findings

This study highlights the complexities and challenges of genomic data analysis in detecting disease-related mutations. The results underscore the need for refined techniques to handle imbalanced data and the importance of comprehensive evaluation metrics. Future work will focus on enhancing model performance through better feature representation and exploring advanced genomic analysis techniques.

5.4 Prospects for Advancements

Possible future developments that result from this thesis in the field of clinical genetics could concentrate on enhancing the precision and accuracy of diagnostic algorithms for genetic diseases. Future studies can enhance our understanding and identification of disease-related mutations by improving current models and incorporating more detailed data. To strengthen diagnostic capabilities, it is crucial to expand the datasets and improve the analytical methods used in modern clinical diagnostics, as they currently rely on limited genetic information.

One possible approach for future research could involve creating sophisticated diagnostic algorithms that utilise AI and ML to examine genetic variations. This would be especially beneficial in situations where conventional approaches are unable to produce definitive

diagnoses. By employing advanced computer models, scientists can improve their understanding of the impact of genetic variants on the development of diseases.

In addition, future studies could prioritise the creation of a robust scoring system that assesses the likelihood of genetic mutations contributing to disease. The proposed scoring system would utilise ML algorithms to combine clinical data, genetic variants, and phenotypic information. This would offer clinicians a robust tool to assist in genetic counselling and personalised therapy planning.

Future developments could also focus on enhancing the integration of genomic data with other types of clinical data, such as imaging and electronic health records, to provide a more holistic view of patient health. Future research can enhance the accuracy and comprehensiveness of diagnostic and prognostic models in clinical genetics by employing an interdisciplinary approach.

6 Conclusions

This study has underscored the complexities inherent in genomic data analysis for detecting disease-related mutations, specifically within the context of hereditary breast cancer. The dataset comprised 7,498 patient records, with 2,449 identified as having breast cancer. Among these, 355 VCF files explicitly listed breast cancer as the diagnosed disease. The reannotation of these VCF files using VEP, and the subsequent mapping of mutation positions with the ClinVar database, ensured that our analysis was precise and aligned with current genomic research standards.

6.1 Key Findings

The feature selection process identified 'IMPACT', 'QUAL', 'DP', 'QD', and 'MAX_AF' as critical for training the XGBoost model. These features were selected to enhance the model's predictive accuracy and interpretability. However, the model's performance indicated significant challenges, particularly due to the imbalanced nature of the dataset. The confusion matrix revealed a high number of true negatives but a low number of true positives, highlighting the model's struggle to detect disease cases.

A key finding is the current limitation of the feature selection, which resulted in a circular logic problem. The critical features identified were necessary because other fields were pruned. This suggests that incorporating more fields and additional data points could improve the model's performance. Future work should focus on expanding the feature set to include more relevant fields to address these challenges and improve the detection of disease-related mutations.

6.2 Model Performance and Limitations

The precision-recall curve provided a nuanced view of the model's performance, showing the trade-offs between precision and recall and highlighting the difficulty in setting appropriate thresholds for imbalanced data. The model's performance metrics were:

- **Mean Precision**: 0.83417
- **Mean Recall**: 0.40091
- **Mean F1 Score**: 0.51713

These metrics underscore the model's limitations in detecting positive cases. Only 7 positive variants were identified with a 0.4 threshold in the validation dataset, demonstrating the need

for more refined techniques to handle imbalanced data. Future experiments should focus on converting more genomic data columns to numeric forms and including additional parameters such as CADD scores, PolyPhen scores, and parallel annotations from multiple databases.

6.3 Strengths and Future Directions

Despite these challenges, the major strength of this work lies in the data handling and unification processes. The dataset itself, which pools hereditary cancers, represents a significant resource for future research. Enriching the phenotypes file with mappings to the ClinVar database and applying filters to improve data quality were crucial steps that enhanced the dataset's robustness.

Future studies should also explore advanced techniques such as natural language processing to match phenotypes to diseases. Additionally, considering the mutation's effect and rarity rather than its specific position may provide better insights into disease-related mutations.

Further developments could include adding more fields and pathogenicity scores, using neural networks, and improving VCF handling instead of relying on TSV intermediaries required for data protection. These advancements would streamline data processing and enhance the overall robustness of genomic analyses.

6.4 Implications for Clinical Genetics

The findings from this study suggest several potential advancements in clinical genetics. Future research could focus on developing dual-pathway diagnostic algorithms that concurrently evaluate monogenic and oligogenic factors contributing to genetic diseases. Such algorithms would be particularly useful when initial testing for a single gene does not yield a definitive diagnosis. Additionally, creating a comprehensive scoring system to assess the probability of oligogenic inheritance versus monogenic inheritance could significantly enhance genetic counseling and tailored therapy planning.

6.5 Summary

In summary, this study highlights the complexities and challenges of genomic data analysis in detecting disease-related mutations. The results underscore the need for refined techniques to handle imbalanced data and the importance of comprehensive evaluation metrics. The work done in data handling and unification sets a strong foundation for future research, and the inclusion of more advanced genomic analysis techniques will likely improve the robustness and accuracy of predictive models in clinical settings.

Bibliography

[1] World Health Organization. Cancer. Web page, May 2023. Overview of cancer, describing its nature, prevalence, and impact globally. Retrieved from https://www.who.int/health-topics/cancer#tab=tab_1.

[2] Z. Ma, C. Y. Woon, C. G. Liu, J. T. Cheng, M. You, G. Sethi, A. L. Wong, P. C. Ho, D. Zhang, P. Ong, L. Wang, and B. C. Goh. Repurposing artemisinin and its derivatives as anticancer drugs: A chance or challenge? *Frontiers in Pharmacology*, 12:828856, 2021.

[3] National Cancer Institute. The genetics of cancer. Website, Aug 2022. Retrieved from https://www.cancer.gov/about-cancer/causes-prevention/genetics.

[4] H. Satam et al. Next-generation sequencing technology: Current trends and advancements. *Biology*, 12(7):997, 2023.

[5] Yanfang Guan, Hong Hu, Yin Peng, Yuhua Gong, Yuting Yi, Libin Shao, Tengfei Liu, Gairui Li, Rongjiao Wang, Pingping Dai, Yves-Jean Bignon, Zhe Xiao, Ling Yang, Feng Mu, Liang Xiao, Zeming Xie, Wenhui Yan, Nan Xu, Dongxian Zhou, and Xin Yi. Detection of inherited mutations for hereditary cancer using target enrichment and next generation sequencing. *Familial Cancer*, 14:9–18, 2015.

[6] Sulev Reisberg. *Developing Computational Solutions for Personalized Medicine*. PhD thesis, University of Tartu, Tartu, Estonia, June 2019. Copiright 2019 by Sulev Reisberg. ISSN: 2613-5906, ISBN: 978-9949-03-091-0 (print), ISBN: 978-9949-03-092-7 (PDF).

[7] Anil Niroula and Mauno Vihinen. How good are pathogenicity predictors in detecting benign variants? *PLoS Computational Biology*, 15(2):e1006481, 2019. License: Creative Commons Attribution License. Funding by Swedish Research Council (Vetenskapsrådet) VR 2015-02510. Data from VariBench database.

[8] William McLaren, Laurent Gil, Sarah E. Hunt, Harpreet Singh Riat, Graham R. S. Ritchie, Anja Thormann, Paul Flicek, and Fiona Cunningham. The ensembl variant effect predictor. *Genome Biology*, 17(1):122, 2016. Open Access.

[9] Ronak Y. Patel, Neethu Shah, Andrew R. Jackson, Rajarshi Ghosh, Piotr Pawliczek, Sameer Paithankar, Aaron Baker, Kevin Riehle, Hailin Chen, Sofia Milosavljevic, Chris Bizon, Shawn Rynearson, Tristan Nelson, Gail P. Jarvik, Heidi L. Rehm, Steven M. Harrison, Danielle Azzariti, Bradford Powell, Larry Babb, Sharon E. Plon, and Aleksandar Milosavljevic. Clingen pathogenicity calculator: a configurable system for assessing pathogenicity of genetic variants. *Genome Medicine*, 9(1):3, 2017. Open Access.

[10] R.M. Ahmad et al. A review of genetic variant databases and machine learning tools for predicting the pathogenicity of breast cancer. *Briefings in Bioinformatics*, 25(1), 2023.

[11] Vatsal Mehra. Snpredict: A machine learning approach for detecting low frequency variants in cancer. Master's thesis, Marquette University, Milwaukee, Wisconsin, 2016. Paper 367. Available at Marquette University e-Publications: `http://epublications.marquette.edu/theses_open/367`.

[12] IBM. What is machine learning?, 2024. Accessed: 26.04.2024.

[13] J.G. Greener, S.M. Kandathil, L. Moffat, and D.T. Jones. A guide to machine learning for biologists. *Nat Rev Mol Cell Biol*, 23(1):40–55, 2022. Epub 2021 Sep 13.

[14] Isha Salian. Supervize me: What's the difference between supervised, unsupervised, semi-supervised and reinforcement learning?, 2018.

[15] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315, 2016.

[16] Alfonso Monaco, Ester Pantaleo, Nicola Amoroso, Antonio Lacalamita, Claudio Lo Giudice, Adriano Fonzino, Bruno Fosso, Ernesto Picardi, Sabina Tangaro, Graziano Pesole, and Roberto Bellotti. A primer on machine learning techniques for genomic applications. *Computational and Structural Biotechnology Journal*, 19:4345–4359, 2021.

[17] Paulo M. do Nascimento, Isac G. Medeiros, Roberto M. Falcão, et al. A decision tree to improve identification of pathogenic mutations in clinical practice. *BMC Medical Informatics and Decision Making*, 20(52), 2020. Received: 30 April 2019; Accepted: 21 February 2020; Published: 10 March 2020.

[18] Bahzad Taha Jijo and Adnan Mohsin Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(1):20–28, 2021.

[19] Manqing Dong, Lina Yao, Xianzhi Wang, Boualem Benatallah, Shuai Zhang, and Quan Sheng. Gradient boosted neural decision forest. *IEEE Transactions on Services Computing*, PP:1–1, Dec 2021.

[20] Carl Kingsford and Steven L. Salzberg. What are decision trees? *Nature Biotechnology*, 26:1011–1013, September 2008.

[21] Tony Duan, Avati Anand, Daisy Yi Ding, Khanh K. Thai, Sanjay Basu, Andrew Ng, and Alejandro Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *JMLR Workshop and Conference Proceedings*, pages 2690–2700, 2020.

[22] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 2013.

[23] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Weilun Wang, Goutam Chakraborty, and Basabi Chakraborty. Predicting the risk of chronic kidney disease (ckd) using machine learning algorithm. *Applied Sciences*, 11(2):202, Dec 2020.

[25] Maxwell Libbrecht and William Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16:321–332, 2015.

[26] Blaise Hanczar, Farida Zehraoui, Tina Issa, and Mathieu Arles. Biological interpretation of deep neural network for phenotype prediction based on gene expression. *BMC Bioinformatics*, 21:501, 2020.

[27] Sebastian Vollmer, Bilal A Mateen, Gergo Bohner, Franz J Király, Rayid Ghani, Pall Jonsson, Sarah Cumbers, Adrian Jonas, Katherine S L McAllister, Puja Myles, David Grainger, Mark Birse, Richard Branson, Karel G M Moons, Gary S Collins, John P A Ioannidis, Chris Holmes, and Harry Hemingway. Machine learning and artificial intelligence research for patient benefit: 20 critical questions on transparency, replicability, ethics, and effectiveness. *BMJ*, 368:l6927, 2020. Accepted: 22 October 2019.

[28] Irene Y. Chen, Emma Pierson, Sherri Rose, Shalmali Joshi, Kadija Ferryman, and Marzyeh Ghassemi. Ethical machine learning in healthcare. *Annual Review of Biomedical Data Science*, 4:123–144, 2021. First published as a Review in Advance on May 6, 2021.

[29] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158, Aug 2011.

[30] Josep F. Abril and Sergi Castellano. Genome annotation. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 195–209. Academic Press, 2019.

[31] National Human Genome Research Institute. Human genome reference sequence, 2024. Updated: May 9, 2024. Text, graphics, videos, illustrations and other information on National Human Genome Research Institute (NHGRI) websites are in the public domain, unless otherwise indicated. You may freely use and distribute this information; however, we request acknowledgement, such as: "Courtesy: National Human Genome Research Institute" with a link back to genome.gov.

[32] Garret A. Tollefson, John Schuster, Frédéric Gelin, et al. Viva (visualization of variants): A VCF file visualization tool. *Sci Rep*, 9(1):12648, 2019. Received: August 14, 2019; Accepted: August 15, 2019; Published: September 2, 2019.

[33] Niloofar Gharani, Gaëlle Calendo, Damir Kusic, et al. Star allele search: a pharmacogenetic annotation database and user-friendly search tool of publicly available 1000 genomes project biospecimens. *BMC Genomics*, 25(1):116, 2024. Received: August 8, 2023; Accepted: January 8, 2024; Published: January 26, 2024.

[34] Mark T. W. Ebbert, Mark E. Wadsworth, Kevin L. Boehme, Kaitlyn L. Hoyt, Aaron R. Sharp, Brendan D. O'Fallon, John S. K. Kauwe, and Perry G. Ridge. Variant tool chest: an improved tool to analyze and manipulate variant call format (vcf) files. *BMC Bioinformatics*, 15(Suppl 7):S12, 2014. From The 10th Annual Biotechnology and Bioinformatics Symposium (BIOT 2013) Provo, UT, USA. 5-6 December 2013.

[35] Hail Team. Hail 0.2.107-2387bb00ceee, 2024. Version 0.2.107-2387bb00ceee. Available online: https://github.com/hail-is/hail/commit/2387bb00ceee. Accessed: 01.04.2024.

[36] *The Variant Call Format Specification*, April 2024. The master version of this document can be found at https://github.com/samtools/hts-specs. This printing is version fe2b48e from that repository, last modified on the date shown above.

[37] Pragati Baheti. Train test validation split: How to & best practices, 2021. Accessed: 26.04.2024.

[38] Name1 Aghan, Syed N. Rahman, Christina W. Agudelo, Alan J. Wein, Jason M. Lazar, Karel Everaert, and Roger R. Dmochowski. Title of the article. *Medicina*, 57:503, 2021. Submission received: April 14, 2021; Revised: May 11, 2021; Accepted: May 14, 2021; Published: May 16, 2021.

[39] Bettina Mieth, Marius Kloft, Juan Rodríguez, et al. Combining multiple hypothesis testing with machine learning increases the statistical power of genome-wide association studies. *Sci Rep*, 6:36671, 2016. Received: May 31, 2016; Accepted: October 6, 2016; Published: November 28, 2016.

[40] Rohit Kundu. F1 score in machine learning: Intro & calculation. *V7 Labs Blog*, December 16 2022.

[41] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PLoS ONE*, 10(3):e0118432, 2015. Received: June 23, 2014; Accepted: January 16, 2015; Published: March 4, 2015.

[42] pandas Development Team. pandas documentation. Documentation, Apr 2024. Version 2.2.2. Retrieved from https://pandas.pydata.org/docs/.

[43] Hail Team. Hail documentation. Documentation, Apr 2024. Retrieved from https://hail.is/docs/0.2/tutorials/01-genome-wide-association-study.html.

[44] Scikit-learn Team. Scikit-learn documentation. Documentation, Apr 2024. Retrieved from https://scikit-learn.org/stable/index.html.

[45] XGBoost Developers. Xgboost documentation. Online, 2022. Revision 82d846bb.

Acknowledgements

I extend my deepest gratitude to MD Villem Pata for his invaluable guidance and profound insights into clinical applications and machine learning techniques, which have been essential to the successful completion of this thesis.

I sincerely thank PhD Hedi Peterson for her support and deep expertise in bioinformatics, which have enriched this research.

I deeply appreciate the chance to work alongside Sander Pajusalu on the ”Oligogenic Inheritance in Genetic Diseases” project at the University of Tartu, which has been a cornerstone of my thesis work.

Lastly, my profound thanks to my family and friends for their unwavering support, and the critical eyes they lent during the proofreading of this thesis.

7 Supplementary Information

This research is part of the *Oligogenic Inheritance in Genetic Diseases* project, running from January 1, 2022, to December 31, 2025. The project is led by Principal Investigator Dr. Sander Pajusalu at the University of Tartu, Faculty of Medicine, Institute of Clinical Medicine, and is funded by the Estonian Research Council under grant number PSG774.

The code developed for this study is published on Github and is publicly available under an open-source license. The code and the corresponding machine learning models can be accessed at the following URL: https://github.com/markus-marandi/XGBoost-Genomics

For inquiries related to the methodology, please contact:

- Markus Marandi, Email: markus.marandi@ut.ee

For inquiries related to the data, please contact:

- Villem Pata, Email: villem.pata@ut.ee

The research is supported by the Estonian Research Council.

7.1 Declaration of Competing Interest

The author and supervisor declare no conflicts of interest regarding this thesis.

8 Addendum

8.1 System Overview

1. Software Setup:

- Operating System: CentOS 9 Stream
- Programming Language: Python 3.7 or newer
- Python Packages:
  - Pandas: Utilised for data manipulation and analysis. [42]
  - Hail: Employed for its powerful handling of genomic data, relying on its scalability for large data sets. [43]
  - Machine Learning Libraries:
    - scikit-learn: Used for machine learning model preparation, including data preprocessing, training, and evaluation. [44]
    - XGBoost: Applied for training gradient boosting models on structured or tabular data. [45]
- Java Runtime Environment (JRE): Required for Hail operation (Java 8 or 11).

2. Hardware Configuration:

- Processor: 16 virtual CPUs (vCPU).
- RAM: 64 GB RAM
- Hard Drive: 500 GB SSD

3. Data Requirements:

- **Phenotypes File Format:** Tab-Separated Values (TSV) file.
- **Content Specifications:**
  - Must include columns for Patient ID, Disease, and Position, organised in this specific order.

  - The Patient ID in the phenotypes file must correspond to the pseudonymised code part of the VCF file names, ensuring accurate linkage between the phenotype data and genomic data.

- **File Encoding:** UTF-8 to ensure compatibility across different data processing scripts and systems.
- **Raw Data Requirements:**
  - The raw data must be in the form of annotated VCF files.
  - These files should be organised within a single directory to streamline data processing and analysis.

8.2 System architecture

8.2.1 Loading and Parsing Phenotypic Data

Objective: Load the phenotypes TSV file and accurately extract disease identification, genetic symbols, and nucleotide expressions.

Implementation:

Data Loading The system reads the TSV file using pandas, with UTF-8 encoding to ensure data integrity.

Data Parsing It iterates through each row, extracting the patient's pseudonymised code (e_code), disease information, genetic symbol (#Symbol), and nucleotide expressions.

Regular Expression Parsing Uses regex to extract genetic identifiers from the nucleotide expressions, such as "NM_####.##:c.*", which are essential for correlating phenotypic and genomic data.

Enhanced Data Structuring Extracted identifiers are sorted and deduplicated. Data is compiled into a dictionary linking each patient code to relevant medical and genetic information.

Including Additional Nucleotide Expressions The additional_nucleotide parameter incorporates extra nucleotide information into the output. This is important because the additional_nucleotide field is preprocessed before analysis to include mappings from the ClinVar database, addressing variations and alternative transcripts from different annotation versions over the years when patients' findings were described.

8.2.2 Processing VCF Files and Generating MatrixTables

Objective:
Convert VCF files into Hail MatrixTables, annotate them with VEP information, and classify them based on disease type and position, storing results in an organised directory structure.

Implementation:

File Discovery and Validation
The system scans a specified source directory or a single file path to gather all VCF files. It validates each file's format and ensures they are either in uncompressed or compressed (gzip) VCF format.

Initialising Hail
Hail is initialised with the GRCh37 reference genome. This step involves setting up appropriate contig recoding to align with the VCF files, simplifying the management of chromosome naming conventions across different genomic data sources.

VCF Processing and Classification:

Each VCF file is processed in sequence. The following steps are executed for each file:

- **Extraction of Patient Information:** The patient code is extracted from the filename, which is used to retrieve corresponding phenotype data (disease and position).
- **Disease Classification:** The disease and position information is used to classify each VCF file into either a positive or negative group for further analysis.
- **File Deduplication:** The system checks for existing output files to avoid reprocessing.

Data Annotation and Filtering
The VCF files are loaded into Hail and the following transformations are applied:

- **Allele Filtering:** Filters out rows with star alleles.
- **Annotation using VEP:** Annotates rows with VEP-derived information such as gene impact, symbol, and allele frequency.
- **Structural Transformations:** Transforms the data by flattening the VEP information and structuring it for efficient querying.

Output Generation
For each processed VCF, the following outputs are generated:

- **MatrixTable Storage:** The annotated MatrixTable is optionally saved in a structured directory based on the disease classification.
- **Tabular Output:** A TSV file containing processed and filtered data is created, which includes critical genomic annotations and patient information.

8.2.3 Data Aggregation for Positive and Negative Groups

Objective:
Aggregate processed data into separate TSV files for positive and negative groups, applying stringent mutation format validation and matching based on phenotypic details.

Implementation:

Regex and Mutation Validation
The system uses regular expressions to verify and match mutation formats within TSV files:

- **Mutation Format Check:** Validates if the mutation string in each file adheres to a predefined format (e.g., "NM_####.##:c.*").
- **Gene and Mutation Extraction:** Extracts gene names and specific mutation details from mutation strings for further processing.

Data Aggregation Logic
Data from individual processed files is aggregated based on their classification into positive or negative groups:

- **File Reading and Processing:**
  - Reads individual TSV files from specified directories for both positive and negative groups.
  - Extracts patient codes from filenames to retrieve corresponding phenotype data.
  - Adds patient codes to DataFrame for tracking and aggregation.
- **Mutation Matching and Aggregation:**
  - Applies regex to match mutations listed in the phenotype data with those in the TSV files.
  - Aggregates files where mutations match the phenotype descriptions, ensuring that gene symbols (if provided) and mutations correspond accurately.
  - Handles cases where mutations need to be partially matched or considered for validation due to high impact.

Output Generation
Creates aggregated TSV files for both positive and negative groups:

- **Negative Group Aggregation:** Combines all data for the negative group into a single TSV file, saving it to the specified directory.
- **Positive Group Aggregation:** Aggregates matched entries for the positive group and saves the compiled data in a designated TSV file.
- **Validation Group Consideration:** Handles entries that do not match expected mutations but are significant enough to warrant further validation.

8.2.4 Preparing Datasets for Machine Learning

Objective:
Prepare and preprocess TSV files for machine learning by selecting relevant columns, encoding features, handling missing values, and scaling numerical data.

Implementation:

Data Loading
Data is loaded from TSV files using pandas, with specific settings to handle NA values and optimize memory usage.

Preprocessing Steps
The preprocessing function includes the following steps to ensure data quality and readiness for machine learning models:

- **Feature Selection:** Filters the dataset to include only relevant columns such as genomic features and quality metrics.
- **Impact Encoding:** Converts categorical descriptions of gene impact into numerical codes to facilitate model interpretation.
- **Chromosome Encoding:** Applies label encoding to chromosome identifiers to convert them into a machine-readable format.
- **Handling Missing Data:**
  - Detects and reports missing values in key features like read depth (DP).
  - Applies median imputation to handle missing values robustly.
- **Feature Scaling:** Standardises features to have zero mean and unit variance, particularly important for algorithms sensitive to feature magnitude.
- **Log Transformation:** Applies a log transformation to allele frequency metrics to normalise the distribution.

Dynamic File Processing
Automates the processing of TSV files within a specified directory:

- **Directory Scanning:** Identifies files intended for machine learning preparation.
- **Data Transformation and Saving:** Processes each file according to the defined preprocessing steps and saves the output to designated subdirectories.

8.2.5 Dividing Data into Training, Testing, and Validation Sets

Objective:
Divide the processed data into balanced training, testing, and validation sets to ensure robust model training and evaluation.

Implementation:

Data Loading
The system dynamically loads processed data from designated sub-directories for positive, negative, and validation groups:

- **Loading Mechanism:** Identifies and aggregates TSV files from each group's scaled directory, appending group labels for easy identification.

Dataset Preparation
Prepares the datasets by ensuring balanced class distribution and splitting data according to predefined proportions:

- **Data Splitting:**
  - Splits the positive group data into training and testing sets with a smaller proportion for testing to maximise training data.

  - Samples an equivalent number of cases from the negative group to maintain balance, ensuring equal representation in training and testing datasets.

- **Validation Set Usage:** Incorporates a separate validation set collected from the validation group for model tuning and unbiased performance estimation.

Balancing and Combining Data
Combines the sampled datasets to form the final training and testing sets:

- **Training Data Assembly:** Concatenates balanced samples from positive and negative training subsets to form a comprehensive training dataset.
- **Testing Data Assembly:** Similarly, combines balanced samples from positive and negative testing subsets.

Output Generation
Saves the prepared datasets to the filesystem for use in machine learning training and evaluation:

- **File Storage:**
  - Saves the training dataset to a file named 'training_set.csv'.
  - Saves the validation dataset to 'validation_set.csv'.
  - Saves the testing dataset to 'testing_set.csv'.

8.2.6 Implementing XGBoost for Model Training and Evaluation

Objective:
Utilise XGBoost to train a machine learning model on the balanced datasets and evaluate its performance using accuracy and feature importance metrics.

Implementation:

Data Preparation
Prepares data for XGBoost training by loading datasets and converting categorical labels into numeric values:

- **Data Loading:** Loads training, validation, and testing datasets from pre-specified file paths.
- **Label Conversion:** Maps group labels to numeric values, facilitating model training.

Model Configuration and Training
Configures and trains an XGBoost model using defined parameters:

- **DMatrix Creation:** Converts data into DMatrix format, which is optimized for XGBoost to improve performance and efficiency.
- **Parameter Setting:** Defines parameters for the XGBoost model (maximum depth, objective function, learning rate, evaluation metrics and random state)

- **Model Training:** Trains the model using the training and validation datasets, tracking performance through each iteration.

Evaluation and Visualisation
Evaluate the model's performance on the testing dataset and visualise important metrics:

- **Accuracy Calculation:** Computes the accuracy of the model predictions against the actual labels in the testing dataset.
- **Feature Importance:** Visualises the importance of different features used in the model to understand their impact on predictions.
- **Decision Tree Visualisation:** Displays the structure of the first ten trees in the model, providing insights into the decision-making process.

Output Generation
Generates and displays output related to the model's performance and key insights:

- **Accuracy Reporting:** Prints the accuracy of the model on the test data.
- **Importance Plotting:** Plots the importance of features using matplotlib, helping identify the most significant predictors.
- **Tree Plotting:** Plots decision trees to visualise how different features and thresholds are used to make predictions.

8.2.7 Model Evaluation and Cross-Validation with XGBoost

Objective:
Evaluate the performance of the XGBoost model using a comprehensive set of metrics, including accuracy, precision, recall, F1 score, and ROC AUC . Further, assess the model's robustness through cross-validation.

Implementation:

Data Loading and Preparation
Loads the prepared datasets and assigns appropriate numeric labels to different groups for model training and testing:

- **Data Loading:** Reads the training, validation, and testing datasets from predefined file paths.
- **Label Assignment:** Maps categorical group labels to numeric codes to facilitate the use in the model.

Model Training and Prediction
Trains an XGBoost classifier on the training data and predicts labels for the test dataset:

- **Feature and Label Separation:** Identifies features and labels within the datasets for use in training and testing.

- **XGBoost Training:** Configures and trains the XGBoost model using defined parameters like max depth and random state.
- **Predictions:** Generates predictions for the test dataset to evaluate model performance.

Performance Metrics
Calculates and prints various performance metrics to assess the effectiveness of the model:

- **Accuracy, Precision, Recall, F1 Score:** Measures the overall accuracy, weighted precision, recall, and F1 score.
- **ROC AUC Score:** Computes the area under the receiver operating characteristic curve to evaluate the trade-offs between true positive rate and false positive rate.
- **Confusion Matrix:** Analyses the true positives, false positives, true negatives, and false negatives to derive sensitivity and statistical power.

Cross-Validation
Performs cross-validation to ensure the model's consistency and reliability across different subsets of the dataset:

- **Scoring Metrics:** Uses multiple scoring metrics such as accuracy, weighted precision, recall, F1, and ROC AUC .
- **Results Analysis:** Provides a summary of cross-validation results, including mean scores and standard deviations for each metric.

A XGBoost Decision Tree

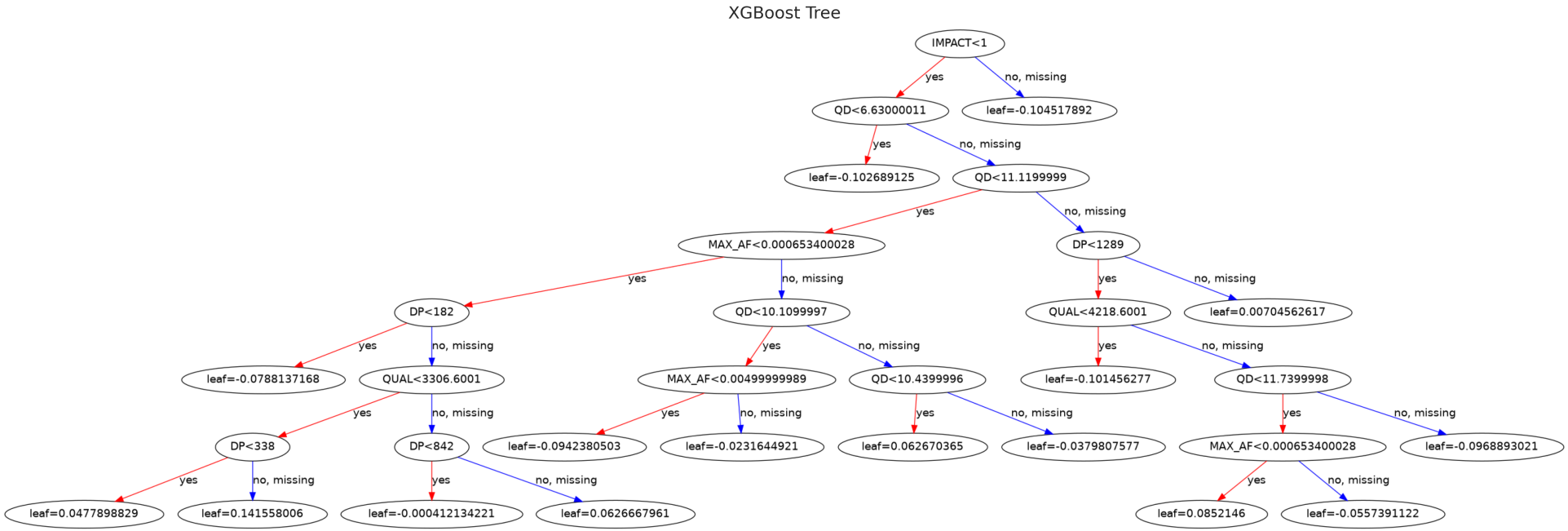

Figure A.1: XGBoost Decision Tree

B Source Code

B.1 Source Code

This section provides the source code used for parsing the phenotypes file and extracting relevant data.

B.1.1 Parsing Phenotypes File

The following Python code parses the phenotypes file to extract e-code, disease, and findings:

```
import pandas as pd
import re

def extract_nm_identifiers(expression):
    if pd.isna(expression):
        return []
    else:
        # Extract NM and other identifiers using regex
        return re.findall(r'(?:LRG_\d+t\d+|NM_\d+\.\d+)(?::c\.[^\s,]+)?',
        expression)

def parse_phenotype_file_HGSV(filepath, additional_nucleotide=True):
    # Load data
    data = pd.read_csv(filepath, sep='\t', encoding='utf-8')

    # Create a dictionary to store extracted data
    extracted_data = {}

    # Iterate over rows
    for index, row in data.iterrows():
        e_code = row['E_code']
        disease = row['Disease']
        symbol = row['#Symbol']
        nucleotide_expression = row['NucleotideExpression']
        additional_nucleotide_expressions = row['Additional_
        NucleotideExpressions']

        # Extract NM identifiers
        nm_identifiers = extract_nm_identifiers(additional_nucleotide_
        expressions)

        # Ensure unique and ordered identifiers to avoid duplication
        unique_identifiers = sorted(set(nm_identifiers), key=lambda x:
```

```
        x.split(':c.')[1] if ':c.' in x else x)

        # Store the extracted data, changing from list to tuple
        if unique_identifiers and additional_nucleotide:
            extracted_data[e_code] = (disease, symbol, nucleotide_
            expression, *unique_identifiers)
        else:
            extracted_data[e_code] = (disease, symbol, nucleotide_
            expression)
    return extracted_data

file_path = '/mnt/sdb/markus-bsc-thesis-data/cleaned-HGSV-phenotype-
globals-tshc-7500.tsv'
phenotype_data = parse_phenotype_file_HGSV(file_path, True)

# Print only the first 20 entries from the extracted data
for i, (e_code, info) in enumerate(phenotype_data.items()):
    if i >= 20:
        break
    print(f"{e_code}: {info}")
```

Listing B.1: Parse phenotypes file

B.1.2 Classify Disease and Data Parsing Functions

The following Python code defines functions for classifying disease and parsing data values:

```
def classify_disease(disease, position):
    if disease == 'RV' and (position != 'NA' and position != 'NEG'):
        return 'positive-group-RV'
    else:
        return 'negative-group'

def parse_empty(text):
    """Converts an empty string to a missing value, otherwise converts to
    integer."""
    return hl.if_else((text == "") | hl.is_missing(text) |
    ~text.matches(r\"\d+"), hl.missing(hl.tint32), hl.int32(text))


def safe_float_parse(s):
    """Parse a float safely, return None if parsing fails due to non-
    numeric string."""
    return hl.if_else(hl.is_defined(s) & s.matches(r'^-?\d*(\.\d+)?$'),
    hl.float64(s), hl.missing(hl.tfloat64))


def parse_to_int32(text):
    return hl.if_else((text == "") | (text == "null"),
    hl.missing(hl.tint32), hl.int32(text))


def parse_to_float64(text):
    # This regex will match numbers including integers, floats, and also
    consider negative values
```

```
    return hl.if_else(hl.is_defined(text) & text.matches(r'^-?\d*\.?\d+
    (?:[Ee][-+]?\d+)?$'), hl.float64(text), hl.missing(hl.tfloat64))


def safe_index(split_list, index, default=None):
    try:
        return split_list[index]
    except IndexError:
        return default
```

Listing B.2: Classify disease and data parsing functions

B.1.3 Parsing and Extracting Data from CSQ Strings

The following Python code demonstrates parsing of CSQ strings to extract relevant genomic data such as impact, gene symbol, HGNC ID, maximum allele frequency, NM accession numbers, mutation notations, and alleles.

```
def parse_csq_final(csq_list, alleles_list):
    parsed_data = []
    for csq, alleles in zip(csq_list, alleles_list):
        # Handle the case where csq is a list and not a string
        csq_str = '|'.join(csq) if isinstance(csq, list) else csq
        # Join list into a single string if it's a list
        csq_str = csq_str.strip('["]') if csq_str else csq_str
        # Ensure csq_str is not None before calling strip
        parts = csq_str.split("|") if csq_str else []

        impact = parts[0] if len(parts) > 0 else "Unknown"
        symbol = parts[1] if len(parts) > 1 else "Unknown"
        hgnc_id = parts[2] if len(parts) > 2 and parts[2].isdigit()
        else None
        max_af = float(parts[3]) if len(parts) > 3 and re.match
        (r'^-?\d*\.?\d+(?:[Ee][-+]?\d+)?$', parts[3]) else 0

        nm_matches = ", ".join(re.findall(r'NM_\d+\.\d+', csq_str))
        c_notations = ", ".join(
            [notation.split(':')[-1] for notation in re.findall
            (r'NM_\d+\.\d+:c\.[\d+-]+[ACTG]>[ACTG]', csq_str)])

        # Handle the case where alleles might also be a list
        alleles_str = '|'.join(alleles) if isinstance(alleles, list)
        else alleles
        alleles_str = alleles_str.strip('[]').replace('"', '').
        replace(' ', '') if alleles_str else alleles_str

        parsed_data.append({
            "IMPACT": impact,
            "SYMBOL": symbol,
            "HGNC_ID": hgnc_id,
            "MAX_AF": max_af,
            "NM_STRING": nm_matches,
            "MUTATIONS": c_notations,
            "alleles": alleles_str
        })
```

```
    return parsed_data
```

Listing B.3: Classify disease and data parsing functions

B.1.4 Processing VCF Files and Annotating Genomic Data

The following Python code demonstrates how to process VCF (Variant Call Format) files, convert them to Hail MatrixTable format, and annotate them with relevant genomic data such as impact, gene symbol, HGNC ID, maximum allele frequency, and more. The code also includes filtering and exporting the processed data to TSV files.

```
def vcfs_to_matrixtable(source, phenotype_data, base_destination,
write=True,
             log_file='/home/markus/gen-toolbox/output/processed_vcfs_
             log.tsv'):
    files = []

    if os.path.isdir(source):
        files = [os.path.join(source, f) for f in os.listdir(source)
    if f.endswith('.vcf') or f.endswith('.vcf.gz')]
    elif os.path.isfile(source) and (source.endswith('.vcf')
    or source.endswith('.vcf.gz')):
        files.append(source)
    else:
        raise ValueError("Invalid path or file type.
        Must be a directory or a VCF file.")

    hl.init(default_reference='GRCh37')  # Initialize Hail
    contig_recoding = {f"chr{i}": str(i) for i in range(1, 23)}
    contig_recoding.update({"chrX": "X", "chrY": "Y"})

    log_entries = []
    processed_vcfs = []

    # Import and annotate files in a loop
    try:
        for vcf in tqdm(files, desc="Processing VCFs"):

            patient_code = os.path.basename(vcf).split('_')[0]
            phenotype_info = phenotype_data.get(patient_code, ("NA", "NA"))

            disease, position = phenotype_info
            group = classify_disease(disease, position)

            destination = os.path.join(base_destination, group)
            destination_path = os.path.join(destination,
            os.path.basename(vcf).replace('.vcf', '.mt'))
            tsv_output_path = os.path.join(destination,
            f"{patient_code}_mtoutput.tsv")

            if os.path.exists(destination_path) or os.path.exists
            (tsv_output_path):
                print(f"Skipping {vcf}, as the output file already exists
                in {destination_path}.")
                log_entries.append({
```

```
                "Timestamp": datetime.now().strftime
                ("%Y-%m-%d %H:%M:%S"),
                "VCF": vcf,
                "Status": "Skipped",
                "Reason": "Output file already exists",
                "Path": destination_path
            })
            continue

        if not os.path.exists(destination):
            os.makedirs(destination)

        mt = hl.import_vcf(vcf, force_bgz=True,
        reference_genome='GRCh37', contig_recoding=contig_recoding,
                           skip_invalid_loci=True)
        mt = mt.filter_rows(mt.alleles[1] != "*")
        # Filter out star alleles
        print(f"Rows before filtering: {mt.count_rows()}")
        mt = mt.filter_rows((hl.len(mt.filters) == 0) |
        mt.filters.contains(hl.literal("PASS")))

        print(f"Rows after filtering PASS: {mt.count_rows()}")
        # Parse CSQ string and annotate rows
        # First, make sure `vep` contains at least one element
        # Annotate rows with the `vep` field
        mt = mt.annotate_rows(
            vep=mt.info.CSQ.map(lambda csq: hl.struct(
                IMPACT=safe_index(csq.split("|"), 0, "Unknown"),
                SYMBOL=safe_index(csq.split("|"), 1, "Unknown"),
                HGNC_ID=hl.if_else(safe_index(csq.split("|"), 2, "").
                matches(r'^-?\d+$'),

                    parse_to_int32(safe_index(csq.split("|"),
                    2)), hl.missing(hl.tint32)),
                MAX_AF=parse_to_float64(safe_index(csq.split("|"),
                3, "0")),
            )),
            CHROM=mt.locus.contig,
            REF=mt.alleles[0],
            ALT=mt.alleles[1]
        )

        # Only keep rows where `vep` is not empty
        mt = mt.filter_rows(hl.len(mt.vep) > 0)
        # Flatten the first struct in the `vep` array directly into
        the row fields
        mt = mt.transmute_rows(
            IMPACT=mt.vep[0].IMPACT,
            SYMBOL=mt.vep[0].SYMBOL,
            HGNC_ID=mt.vep[0].HGNC_ID,
            MAX_AF=mt.vep[0].MAX_AF
        )
        #mt.info.CSQ.show()
        # Define the TSV output path using the patient code and group
        df = mt.rows().flatten().to_pandas()
        # Process and filter necessary fields with `parse_csq_final`
        processed_data = parse_csq_final(df['info.CSQ'].tolist(),
        df['alleles'].tolist())
        filtered_df = pd.DataFrame(processed_data)
```

```
            # Add the columns to the DataFrame
            filtered_df['CHROM'] = df['CHROM']
            filtered_df['REF'] = df['alleles'].apply(lambda x: x[0])
            filtered_df['ALT'] = df['alleles'].apply(lambda x: x[1]
            if len(x) > 1 else None)
            filtered_df['QUAL'] = df['qual']
            for col in df.columns:
                if col.startswith('info.') and not col.endswith('CSQ'):
                    filtered_df[col.split('.')[1]] = df[col].apply(
                        lambda x: x[0] if isinstance(x, list) and
                        len(x) > 0 else x)

            if write:
                # Define the TSV output path and export
                tsv_output_path = os.path.join(destination,
                f"{patient_code}_mtoutput.tsv")
                filtered_df.to_csv(tsv_output_path, sep='\t',
                index=False)
                processed_vcfs.append({'vcf': vcf, 'mt_path':
                destination_path})
                log_entries.append({
                    "Timestamp": datetime.now().strftime
                    ("%Y-%m-%d %H:%M:%S"),
                    "VCF": vcf,
                    "Status": "Processed",
                    "Reason": "Successfully processed and saved",
                    "Path": tsv_output_path
                })


    finally:
        hl.stop()  # Stop Hail context when done
        log_df = pd.DataFrame(log_entries)
        log_df.to_csv(log_file, sep='\t', index=False)

    return processed_vcfs, log_entries


# Example usage:
VEP_CONFIG_PATH = '/home/markus/gen-toolbox/src/config/vep_settings.json'
SOURCE_DIR = '/mnt/sdb/TSHC_data_5k'
base_destination_directory = '/mnt/sdb/markus-bsc-thesis-data'
phenotype_data = parse_phenotype_file('/mnt/sdb/phenotype-globals-tshc-
7500.tsv') # file for one result per e-code
log_entries = vcfs_to_matrixtable(SOURCE_DIR, phenotype_data,
base_destination_directory, write=True)
print(log_entries)
```

Listing B.4: Classify disease and data parsing functions

B.1.5 Aggregating Data to Create Positive and Negative Groups

The following Python code demonstrates how to aggregate data from multiple TSV files to create comprehensive positive and negative group tables. It includes regex search functionalities to identify valid mutations, extract gene and mutation information, and handle data accordingly.

```
def valid_mutation_format(mutation):
    """ Check if the mutation string format is valid.
    Only proceed if mutation is a string. """
    if not isinstance(mutation, str):
        return False
    pattern = re.compile(r'NM_\d+\.\d+:c.*', re.IGNORECASE)
    return bool(pattern.match(mutation))

def extract_gene_and_mutation(mutation_info):
    try:
        gene_name, mutation_detail = mutation_info.split(':')mutation_
        pattern = re.compile(r'c\.\d+[+-]?\d*(?:[GATC]>[GATC]|del[ACTG]+
        |ins[ACTG]+|\d+_\d+del|\d+_\d+ins[ACTG]+)', re.IGNORECASE)
        matches = mutation_pattern.search(mutation_detail.strip())
        # strip to remove any leading/trailing spaces
        if matches:
            return gene_name, matches.group()
        return gene_name, None
    except ValueError:
        return None, None

def aggregate_tsv_files(base_path, phenotype_dict,
process_negative_group=True, process_positive_group=True, symbol=True,
mutation=True):
    negative_group_path = os.path.join(base_path, 'negative-group')
    positive_group_path = os.path.join(base_path, 'positive-group-RV')
    neg_dfs = []
    pos_dfs = []
    val_dfs = []

    if process_negative_group:
        neg_files = os.listdir(negative_group_path)
        for file in tqdm(neg_files, desc="Processing Negative Group"):
            file_path = os.path.join(negative_group_path, file)
            df = pd.read_csv(file_path, sep='\t')
            e_code = file.split('_')[0]
            df['E_code'] = e_code
            neg_dfs.append(df)

    if process_positive_group:
        pos_files = os.listdir(positive_group_path)
        for file in tqdm(pos_files, desc="Processing Positive Group"):
            file_path = os.path.join(positive_group_path, file)
            df = pd.read_csv(file_path, sep='\t', low_memory=False)
            e_code = file.split('_')[0]
            df['E_code'] = e_code

            if e_code in phenotype_dict:
                _, _symbol, *mutations = phenotype_dict[e_code]
                mutations = [m for m in mutations if
                valid_mutation_format(m)]
                if mutations:  # Only proceed if there are valid mutations
                  subtracted_regex = None
                    if not mutation: # exclude mutations
                      mutation_regex = '|'.join(re.escape(m.split(":")[0])
                      for m in mutations)
                    else:
                        mutation_regex = '|'.join(re.escape(m)
```

```
                    for m in mutations)
                    subtracted_mutation_regex = '|'.join(re.escape
                    (m[:-2]) for m in mutations)
                    # Pattern after removing last string from mutation
                    subtracted_regex = re.compile(subtracted_mutation_
                    regex, re.IGNORECASE)


                regex = re.compile(mutation_regex, re.IGNORECASE)

                mask = df['MUTATIONS'].astype(str).apply(lambda
                x: subtracted_regex.search(x) is not None
                if regex.search(x) is None and mutation else regex.
                search(x) is not None)
                df["nucleotide_expression"] = df["NM_STRING"] +
                ":" + df["MUTATIONS"] # Combine expression and mutation
                df["secondary_matched"] = False
# Field to track if mutation matcches the subtracted regex pattern

                if symbol and _symbol:
                # If symbol is true and e_code symbol is not null
                   df["matched"] = (df["nucleotide_expression"].
                   str.contains(regex.pattern, na=False) &
                   (df["SYMBOL"] == _symbol))
                   if subtracted_regex:
                       df["secondary_matched"] = (df["nucleotide_
               expression"].str.contains(subtracted_regex.pattern,
            na=False) & (df["SYMBOL"] == _symbol))
                else:
                    df["matched"] = df["nucleotide_expression"].
                    astype(str).apply(lambda x: regex.search(x)
                    is not None)
                    if subtracted_regex:
                       df["secondary_matched"] = df["nucleotide_
                       expression"].astype(str).apply(lambda
                       x: regex.search(x) is not None)

                target_rows = df[(df["matched"] |
                ((df["matched"] == False) & (df["secondary_matched"]) &
                (df["IMPACT"] == "HIGH")))]
                # rows that match the expression
                if not target_rows.empty:
                    df.loc[mask, 'E_code'] = e_code
                    pos_dfs.append(target_rows)
                else:
                    # If no matching mutation is found, consider
                    it for validation group
                    val_dfs.append(df)

# Aggregate and save DataFrames as done previously
# Combine all DataFrames for the negative group and save
if neg_dfs:
    negative_agg_df = pd.concat(neg_dfs, ignore_index=True)
    #negative_agg_df.drop(["nucleotide_expression", "matched"],
    inplace=True, axis=1)
    negative_agg_df.to_csv(os.path.join(base_path,
    'negative_group_aggregated.tsv'), sep='\t', index=False)
    print("Negative group data saved successfully.")
```

```
    # Combine all DataFrames for the positive group and save if there
    are any
    if pos_dfs:
        positive_agg_df = pd.concat(pos_dfs, ignore_index=True)
        positive_agg_df.drop(["nucleotide_expression", "matched",
        "secondary_matched"], inplace=True, axis=1)
        positive_agg_df.to_csv(os.path.join(base_path, 'positive_group_
        aggregated.tsv'), sep='\t', index=False)
        print("Positive group data saved successfully.")

    if val_dfs:
        print("Aga siia")
        validation_agg_df = pd.concat(val_dfs, ignore_index=True)
        validation_agg_df.drop(["nucleotide_expression", "matched",
        "secondary_matched"], inplace=True, axis=1)
        validation_agg_df.to_csv(os.path.join(base_path,
        'validation_group_aggregated.tsv'), sep='\t', index=False)
        print("Validation group data saved successfully.")

    else:
        print("No data for positive group or all data were non-matching.")


def regex_search(symbol=True, mutation=True, additional_nucleotide_
expressions=True):
    # Assuming phenotype_data is loaded properly
    phenotype_data = parse_phenotype_file_HGSV('/mnt/sdb/markus-bsc-thesis-
    data/cleaned-HGSV-phenotype-globals-tshc-7500.tsv',
    additional_nucleotide_expressions)
    aggregate_tsv_files('/mnt/sdb/markus-bsc-thesis-data/', phenotype_data,
    process_negative_group=True, process_positive_group=True,
    symbol=symbol, mutation=mutation)

regex_search(symbol=True, mutation=True, additional_nucleotide_
expressions=True)
```

Listing B.5: Aggregating data to create positive and negative group tables

B.1.6 Preprocessing Datasets for Machine Learning

The following Python code demonstrates how to preprocess datasets by selecting the most relevant features and performing necessary imputation for missing values. This preprocessing step ensures the data is ready for machine learning without scaling.

```
# Load data
def load_data(file_path):
    df = pd.read_csv(file_path, sep='\t', na_values=['NA', 'null', ''],
    low_memory=False)
    return df

# Preprocessing function
def preprocess_data(df):
    # Creating a copy to avoid SettingWithCopyWarning
    df = df.copy()
```

```
    # Selecting the relevant columns
    relevant_columns = ['IMPACT', 'QUAL', 'DP', 'QD', 'MAX_AF']
    df = df[relevant_columns]

    # Impact mapping using .loc to avoid SettingWithCopyWarning
    impact_mapping = {'HIGH': 0, 'MODERATE': 1, 'LOW': 2, 'MODIFIER': 3}
    df.loc[:, 'IMPACT'] = df['IMPACT'].map(impact_mapping)

    # Handle missing values for 'DP' (Depth of coverage)
    if df['DP'].isnull().any():
        print("NaN values found in DP. Imputing with median.")
        dp_imputer = SimpleImputer(strategy='median')
        df['DP'] = dp_imputer.fit_transform(df['DP'].values.reshape(-1, 1))

    # Handle missing values for 'MAX_AF' (Maximum Allele Frequency)
    if df['MAX_AF'].isnull().any():
        max_af_imputer = SimpleImputer(strategy='median')
        df['MAX_AF'] = max_af_imputer.fit_transform(df[['MAX_AF']])

    # Handling numeric columns 'QUAL' and 'QD'
    numeric_cols = ['QUAL', 'QD']
    for col in numeric_cols:
        if df[col].isnull().any():
            print(f"NaN values found in {col}. Imputing with median.")
            imputer = SimpleImputer(strategy='median')
            df[col] = imputer.fit_transform(df[[col]])

    return df

def process_files_dynamically(base_directory):
    for file_name in os.listdir(base_directory):
        if file_name.endswith('_aggregated.tsv'):
            group_subfolder = None
            if 'positive' in file_name:
                group_subfolder = 'positive_group'
            elif 'negative' in file_name:
                group_subfolder = 'negative_group'
            elif 'validation' in file_name:
                group_subfolder = 'validation_group'

            if group_subfolder:
                # Create the group directory if it doesn't exist
                group_directory = os.path.join(base_directory,
                group_subfolder)
                os.makedirs(group_directory, exist_ok=True)

                # Create the scaled subdirectory within the group directory
                scaled_directory = os.path.join(group_directory, 'scaled')
                os.makedirs(scaled_directory, exist_ok=True)

                # Define input and output file paths
                input_file_path = os.path.join(base_directory, file_name)
                output_file_name = 'ML_prepped_' + file_name
                output_file_path = os.path.join(scaled_directory,
                output_file_name)
                process_and_save_file(input_file_path, output_file_path)

def process_and_save_file(input_file_path, output_file_path):
    df = load_data(input_file_path)
```

```
    df_processed = preprocess_data(df)
    df_processed.to_csv(output_file_path, index=False, sep='\t')
    print(f"Data processed and saved to {output_file_path}")

# Example call to the function
base_directory = '/mnt/sdb/markus-bsc-thesis-data/machine-learning'
process_files_dynamically(base_directory)
```

Listing B.6: Preprocessing datasets for machine learning

B.1.7 Dividing Data into Training, Testing, and Validation Sets

The following Python code demonstrates how to divide the data into training, testing, and validation sets using positive, negative, and validation groups. It also includes shuffling and saving the datasets to ensure random distribution.

```
def load_data(base_dir, group_name):
    tsv_dir = os.path.join(base_dir, group_name, 'scaled')
    print(f"Loading data from {tsv_dir}")
    files = glob(os.path.join(tsv_dir, "*.tsv"))
    data_list = [pd.read_csv(file, sep='\t') for file in files]
    if data_list:
        data = pd.concat(data_list)
        data['group'] = group_name
    else:
        data = pd.DataFrame()
    return data

def load_all_groups(base_directory):
    groups = ["positive_group", "negative_group", "validation_group"]
    data_frames = {}
    for group in groups:
        data_frames[group] = load_data(base_directory, group)
    return data_frames

def prepare_datasets(data_frames):
    data_positive = data_frames["positive_group"]
    data_negative = data_frames["negative_group"]
    validation_data = data_frames["validation_group"]
    # Using provided validation data directly
    validation_data.to_csv(f"{base_directory}/validation_set.csv",
    index=False)

    # Combine positive and negative data, excluding validation data
    combined_data = pd.concat([data_positive, data_negative])
    total_positives = combined_data[combined_data['group'] ==
    'positive_group'].shape[0]
    total_negatives = combined_data[combined_data['group'] ==
    'negative_group'].shape[0]
    print(f"Combined data count: {combined_data.shape[0]} rows
    (Positive: {total_positives}, Negative: {total_negatives})")

    # Splitting combined data into training and testing sets (70/30 split)
    train_data, test_data = train_test_split(combined_data, test_size=0.3,
    random_state=42)
```

```
    # Checking if total rows in train and test match the combined data rows
    total_train_test_rows = train_data.shape[0] + test_data.shape[0]
    print(f"Total rows in train + test: {total_train_test_rows} rows.
    Matches combined data: {total_train_test_rows == combined_data.
    shape[0]}")

    # Separating positive and negative data within training and testing
    sets
    train_pos = train_data[train_data['group'] == 'positive_group']
    train_neg = train_data[train_data['group'] == 'negative_group']
    test_pos = test_data[test_data['group'] == 'positive_group']
    test_neg = test_data[test_data['group'] == 'negative_group']

    # Calculating distributions
    train_pos_pct = (train_pos.shape[0] / train_data.shape[0]) * 100
    train_neg_pct = (train_neg.shape[0] / train_data.shape[0]) * 100
    test_pos_pct = (test_pos.shape[0] / test_data.shape[0]) * 100
    test_neg_pct = (test_neg.shape[0] / test_data.shape[0]) * 100

    print(f"Distribution in training set Positive: {train_pos_pct:.2f}%,
    Negative: {train_neg_pct:.2f}%")
    print(f"Distribution in testing set Positive: {test_pos_pct:.2f}%,
    Negative: {test_neg_pct:.2f}%")

    # Ensuring both sets contain positive and negative data (following the
    natural distribution)
    train_data = pd.concat([train_pos, train_neg])
    test_data = pd.concat([test_pos, test_neg])

    print(f"Final Training set: {train_data.shape[0]} rows (Positive:
    {train_pos.shape[0]}, Negative: {train_neg.shape[0]})")
    print(f"Final Testing set: {test_data.shape[0]} rows (Positive:
    {test_pos.shape[0]}, Negative: {test_neg.shape[0]})")
    print(f"Validation set: {validation_data.shape[0]} rows")

    return train_data, test_data

def shuffle_and_save(data, file_path):
    # Shuffle data
    column_headers = data.columns
    shuffled_data = data.sample(frac=1, random_state=42)
    # Save to TSV
    shuffled_data.to_csv(file_path, index=False, header=column_headers)

base_directory = '/mnt/sdb/markus-bsc-thesis-data/machine-learning'
data_frames = load_all_groups(base_directory)
training_data, testing_data = prepare_datasets(data_frames)

shuffle_and_save(training_data, f"{base_directory}/training_set.csv")
shuffle_and_save(testing_data, f"{base_directory}/testing_set.csv")
```

Listing B.7: Dividing data into training, testing, and validation sets

B.1.8 Training the XGBoost Model

The following Python code demonstrates how to train an XGBoost model using the preprocessed training dataset. It includes loading the data, preparing the DMatrix objects for XGBoost, defining the model parameters, training the model, and evaluating its performance. Additionally, it plots feature importance and a decision tree from the trained model.

```
# Define the base directory and paths
base_directory = '/mnt/sdb/markus-bsc-thesis-data/machine-learning'
training_file = f"{base_directory}/training_set.csv"
validation_file = f"{base_directory}/validation_set.csv"
testing_file = f"{base_directory}/testing_set.csv"

# Function to load and prepare the data
def load_data(file_path):
    data = pd.read_csv(file_path)
    # Convert group to numeric labels: assuming 'positive_group' as 1,
    'negative_group' as 0
    label_mapping = {'positive_group': 1, 'negative_group': 0,
    'validation_group': 2}  # Update as per actual data
    data['label'] = data['group'].map(label_mapping)
    data.drop(['group'], axis=1, inplace=True)
    # Remove 'group' column from features
    return data

# Load datasets
training_data = load_data(training_file)
validation_data = load_data(validation_file)
testing_data = load_data(testing_file)

# Prepare XGBoost DMatrices
dtrain = xgb.DMatrix(training_data.drop('label', axis=1),
label=training_data['label'])
dval = xgb.DMatrix(validation_data.drop('label', axis=1),
label=validation_data['label'])
dtest = xgb.DMatrix(testing_data.drop('label', axis=1),
label=testing_data['label'])

# Define XGBoost model parameters
params = {
    'max_depth': 7,
    'objective': 'binary:logistic',
    'eta': 0.1,
    'eval_metric': 'logloss',
    'random_state': 42
}
num_rounds = 100

# Initialize an empty dictionary to store evaluation results
evals_result = {}

# Initialize lists to store classification error for each dataset
train_error = []
val_error = []
test_error = []

# Train the model
```

```
model = xgb.train(params, dtrain, num_rounds, evals=[(dtrain, 'train'),
(dtest, 'test')], evals_result=evals_result)

# Predictions and evaluation
predictions_proba = model.predict(dtest)
predictions = [1 if p >= 0.4 else 0 for p in predictions_proba]
accuracy = accuracy_score(testing_data['label'], predictions)
precision = precision_score(testing_data['label'], predictions)
recall = recall_score(testing_data['label'], predictions)
f1 = f1_score(testing_data['label'], predictions)
logloss = log_loss(testing_data['label'], predictions_proba)
roc_auc = roc_auc_score(testing_data['label'], predictions_proba)
# Calculate ROC AUC
pr_auc = average_precision_score(testing_data['label'], predictions_proba)
# Calculate PR AUC

print(f"Test Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Log Loss: {logloss}")
print(f"ROC AUC: {roc_auc}")
print(f"PR AUC: {pr_auc}")

# Plotting feature importance and decision tree
xgb.plot_importance(model)
fig, ax = plt.subplots(figsize=(20, 10), dpi=300)
xgb.plot_tree(model, num_trees=10, ax=ax)
plt.title('XGBoost Tree')
plt.savefig('xgb_tree_high_res.png', dpi=300)
plt.show()
```

Listing B.8: Training the XGBoost model

B.1.9 Enriching phenotypes file with ClinVar database

The following Python code demonstrates how to enrich a phenotypes file with additional data from the ClinVar database. It includes loading and modifying the TSV file, extracting relevant information, and matching it with the ClinVar variation data to retrieve the AlleleID for further analysis.

```
import pandas as pd

def create_modified_tsv(input_filepath, output_filepath):
    # Step 1: Load the existing TSV file
    df = pd.read_csv(input_filepath, delimiter='\t', encoding='ISO-8859-1',
    header=None, names=['E_code', 'Disease', 'Result'])
    df['Result'] = df['Result'].fillna('NA')  # Replace any NaNs in
    'Result' with 'NA'
    print("Original DataFrame loaded with shape:", df.shape)

    # Step 2: Extract and format the NM string with position
    def format_nm_string(result):
        try:
            # Extracting up to the comma
```

```
            result_up_to_comma = result.split(',')[0]
            # Extracting the NM identifier part before the parenthesis and
            mutation info after the colon
            nm_part = result_up_to_comma.split(':')[0]
            # This splits the NM_000051.4(ATM)
            mutation_part = result_up_to_comma.split(':')[1]
            # This splits the mutation part
            nm_identifier = nm_part.split('(')[0]
            # Removes any gene name and parenthesis
            formatted_result = nm_identifier + ':' + mutation_part
            # Combining them
            return formatted_result
        except Exception as e:
            return "Parse Error\"

    df['Formatted_Result'] = df['Result'].apply(format_nm_string)
    print("Formatted Result extraction completed.")

    # Step 3: Save the new TSV file
    df.to_csv(output_filepath, sep='\t', index=False, encoding='UTF-8')
    print("Modified DataFrame saved to:", output_filepath)
    return df

def enrich_tsv_with_allele_id(main_tsv_path, variation_txt_path,
output_tsv_path):
    # Load the main TSV file
    df = pd.read_csv(main_tsv_path, delimiter='\t', low_memory=False)
    print("Main DataFrame loaded with shape:", df.shape)

    # Load the variation TXT file
    variation_df = pd.read_csv(variation_txt_path, delimiter='\t',
    low_memory=False)
    print("Variation DataFrame loaded with shape:", variation_df.shape)

    # Filter main DataFrame for relevant rows based on Disease and
    Formatted_Result
    # Using the non-null index ensures compatibility
    df['Filtered'] = df.apply(lambda row: row['Disease'] == 'RV' and
    row['Formatted_Result'] not in ['NA', 'NEG'], axis=1)

    # Normalize NucleotideExpression in the variations DataFrame for
    accurate matching
    variation_df['Key_Expression'] = variation_df['NucleotideExpression'].
    apply(lambda x: re.sub(r'[^\w:>.]', '', x) if pd.notnull(x) else '')

    # Filter variations DataFrame to include only "coding" type entries
    coding_variations = variation_df[variation_df['Type'] == 'coding'].
    set_index('Key_Expression')['AlleleID']

    # Define a function to fetch AlleleID based on the cleaned-up
    Formatted_Result
    def get_allele_id(formatted_result):
        key = re.sub(r'[^\w:>.]', '', formatted_result)
        # Clean the formatted_result for matching
        return coding_variations.get(key, '')

    # Apply function to get AlleleID only for filtered rows
    df['AlleleID'] = df[df['Filtered']]['Formatted_Result'].
    apply(get_allele_id)
```

```
    # Save the enriched DataFrame to a new TSV file
    df.to_csv(output_tsv_path, sep='\t', index=False, encoding='UTF-8')
    print("Enriched DataFrame saved to:", output_tsv_path)
    return df

output_df = enrich_tsv_with_allele_id(
    '/mnt/sdb/markus-bsc-thesis-data/HGSV-phenotype-globals-tshc-7500.tsv',
    '/mnt/sdb/markus-bsc-thesis-data/hgvs4variation.txt',
    '/mnt/sdb/markus-bsc-thesis-data/enriched-HGSV-phenotype-globals-tshc-
    7500.tsv'
)
```

Listing B.9: Enriching phenotypes file with ClinVar database

Licence

I, Markus Marandi

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Machine Learning Framework for Classification of Potential Hereditary Cancers

   supervised by Villem Pata

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Markus Marandi 20/05/2024