Pipeline for predicting variant impact using CADD

NAME STUDENT: Seyan Hu

REG.NR.: 1128566

Master thesis Animal Breeding and Genomics (exam code ABG80436)

March 2023

Supervisors: Martijn Derks, prof.dr.ir. Dick de Ridder

Thesis: Animal Breeding and Genomics

WAGENINGEN
UNIVERSITY & RESEARCH

Table of Contents

Abstract

Combined Annotation Dependent Depletion (CADD) is a simple and powerful Machine Learning based method that is able to predict the impact of genetic variants in coding regions as well as non-coding regions across the genome. At its core, it is a classifier that is trained to distinguish between two classes of variants: derived variants (class 1, assumed to be benign) and simulated variants (class 0, assumed to be deleterious). After prediction, it transforms the posterior likelihood (the RAW score) of class 0 to a PHRED-like score (the CADD score), the higher the score the more likely the variant is deleterious. This method is used in various fields of study for all kinds of different species, such as breeding strategies in agriculture or animal husbandry.

Since previous CADD implementations were outdated, incohesive and inflexible, a new implementation of the CADD pipeline has been developed in Python v3.6. The code was modularized into five snakefiles with Snakemake and hard-coded parameters were replaced by command-line options within the snakefiles.

The resulting new implementation is more cohesive and flexible, and only command-line options and different genomic annotations are needed when performed for different species. Overall, the new implementation of the CADD pipeline works, as genome-wide CADD scores are generated for mouse chromosome 19 (GRCm39).

1. Introduction

In biology, understanding the relationship between an organism's genotype and its phenotype is a major challenge, since this is influenced by complex interactions between numerous genes as well as with the environment [1]. This genotype-phenotype relationship plays a key role in many areas of research, such as medicine or agriculture and animal husbandry. One method to better understand this relationship is to predict the influence or impact of genetic variants on the phenotype with Machine Learning (ML), i.e. models that learn to make predictions based on given example data (features and labels) [2].

Predicting the impact of variants in protein-coding elements is widely studied [3], as variants can immediately affect the protein function and thereby potentially the phenotype. These mutations in protein-coding elements have a greater chance of directly affecting the organism [4]. However, non protein-coding elements are also relevant, since these may regulate gene expression and are assumed to have a larger impact on the organism's phenotype than changes in protein-coding elements.

Previous variant effect prediction strategies, such as PolyPhen [5], SIFT [5], REVEL [6], MutationTaster [3], Provean [3] or MutationAssessor [3], use annotations (various genomics-derived measures) to prioritize causal variants in protein-coding elements. These strategies are limited as they use a single information type (e.g. only conservation scores, et cetera) and are only able to predict the effect of variants that affect the protein sequence (e.g. missense mutations, nonsense mutations, et cetera) [5]. Other strategies such as CADD [7], Basset [8] or DeepSEA [9] are able to predict the impact of variants in protein-coding elements as well as non protein-coding elements.

In this study, the focus is on CADD, since CADD is one of the first methods for genome-wide prediction of the impact of variants. CADD is open source and simple, therefore it was used as a base for the development of many prediction-based strategies for other species. At its core, CADD is a classifier that is trained to distinguish between two classes of variants: derived variants and simulated variants. Derived variants are generated by sampling alleles that have been fixed in the species since their last ancestor; fixation is assumed to be caused by selective pressure (Figure 1). These are mostly depleted in deleterious variants and are therefore considered proxy-benign or proxy-neutral [10]. Simulated variants are generated *de novo*, based on the inferred substitution rates. These variants are not observed and may thus be enriched for deleterious variants [11]. Hence they are considered proxy-deleterious [12].

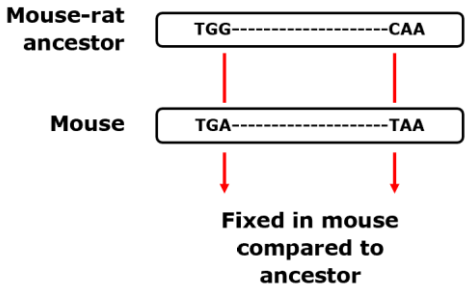

Figure 1: Example of how derived variants are inferred by comparing the ancestral sequence with the current genome sequence. In this example, a mouse-rat ancestor sequence is compared to a mouse sequence. Derived variants are determined by sampling alleles in the mouse sequence that differs in comparison to the alleles in the ancestor.

CADD is currently used in human research, as well as research on animal species (e.g. livestock). It was first developed for the human genome (hCADD) to determine deleteriousness for single-nucleotide variants, and short insertion/deletions [13]. For animal species, specifically livestock,

CADD is used to identify genome-wide variants with relatively high importance (deleterious phenotype-changing variants), as these deleterious variants do not necessarily affect the organism negatively and might have a positive impact on the organism. These variants could therefore be used for breeding strategies in agriculture or animal husbandry [12]. After the development of hCADD, a mouse CADD (mCADD) model was developed to create the first working CADD for animal species, but in the end genome-wide mCADD scores were never generated [2]. Subsequently, CADD was also developed for various livestock species, such as pig (pCADD) [1] [11] and chicken (chCADD) [12] [4]. Different kinds of machine learning models were used for CADD: a Support Vector Machine hCADD and Logistic Regression mCADD, pCADD and chCADD.

The latest CADD pipeline that is developed by Christian Groß [12] for livestock species is incohesive as it consists of many scripts that need to be run manually individually. It is also inflexible, as it is difficult to update existing CADD scores that were generated with outdated annotations (features), add new annotations (features) to the model or generate CADD scores for different species. Pipeline flexibility is important, since species alignments or annotations are constantly being updated.

As a result of these limitations, CADD cannot be easily performed on different species without many manual adjustments in a large number of scripts. Therefore, the aim of this study is to develop a more cohesive and flexible CADD pipeline, that is able to run the consecutive scripts with fewer steps and that can be applied to new species with fewer manual adjustments. This new CADD pipeline can contribute to upcoming studies regarding the relationship between genotype and phenotype and will make it easier for biologists to use the CADD approach in specific studies.

2. Materials & Methods

2.1 CADD overview

CADD is a classifier that makes use of multiple annotations and species-specific evolutionary models that capture signals of natural selection (or selective breeding) across many generations [2], eventually integrating them into a single metric called the CADD score [13]. These multiple annotations consist of various genomics-derived measures (e.g. conservation scores, consequences on the encoding amino acid, affected genes/transcripts, et cetera). The CADD model was trained to distinguish between two classes of variants: derived variants (assumed to be benign) and simulated variants (assumed to be potentially deleterious). The model was trained with the gathered annotations as features and the classes of the generated variants as labels, to fit the model parameters [11]. CADD outputs a RAW score that was afterwards converted to a PHRED-scaled score (the CADD score).

The higher the score the more likely the variant is deleterious. A CADD score of 10 indicates that the raw score is within the top 10% of all reference variants, a score of 20 indicates a top 1% variant, et cetera [10]. Genome-wide CADD scores are usually generated for all possible variants at every position in the genome.

2.2 Starting point

A new general CADD implementation was developed that is able to run consecutive scripts with fewer manual (by hand) adjustments and can be applied to new species (Figure 2). For the development of a more cohesive and flexible CADD pipeline, the latest instalment of the livestock CADD (chCADD (chicken CADD) by Christian Groß [12]) was used as a base, since it contained the most up-to-date scripts. A pipeline is here defined as consecutive processes (that execute tasks) connected in series, where the output of one process is the input of the next.

chCADD currently runs on an old version of Python (v2.7) [12] and needs much manual user input to work properly. This is caused by a large number of scripts that contain hard-coded data or paths, i.e. embedded into the code itself.

For the development of the new pipeline implementation, the mouse species GRCm39 was used since it is a model species for studying diseases/health and therefore well-studied, with a large amount of publicly available genomic annotation data [2]. Since this project focuses on a more general CADD implementation that was developed to be more cohesive and flexible in comparison to the previous pipeline, the eventual (mouse) mCADD model will not be a one-to-one recreation of the previous mCADD model by Christian Groß [2], since it only uses annotations that are by default available for a large number of different species.

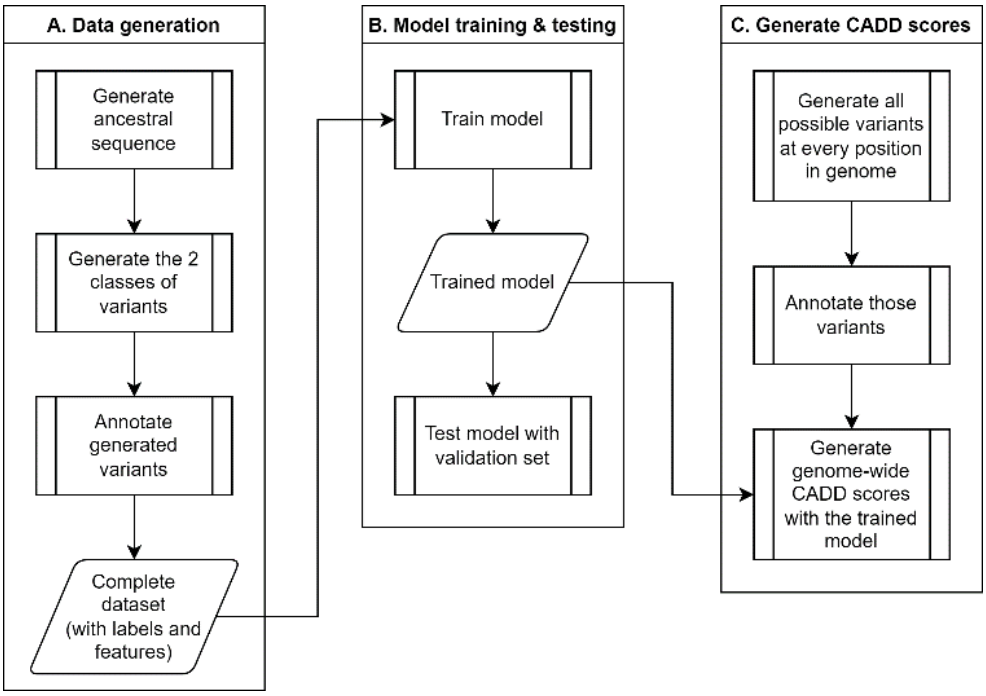

Figure 2: Overview of the pipeline for the prediction of the impact of variants using CADD. The pipeline is divided into three distinct sections: the generation of the training data (A), the training/testing of the model (B) and the generation of genome-wide CADD scores (C).

2.3 General implementations

The new pipeline was developed in a Conda v4.12.0 environment [14]. Anaconda was used to install packages and tools that were used in the previous CADD implementation. To make the CADD pipeline more cohesive and flexible, the latest chCADD was first adjusted to work with a newer version of Python. Specifically, v3.6.13 [15], since v3.6 is one of the more widely used subversions of Python3 that does not cause conflicts between the Python packages that were already in use by chCADD.

Concurrently, the code was modularized (e.g. generation of ancestral sequence, generation of derived/simulated variants, generation of annotations, model training & validation, generation of genome-wide CADD scores, see Figure 2), to make the pipeline easier to use and the development more manageable. This means that the processes in the pipeline were separated into modules/parts based on their functionality. For each modularized part, Snakemake wrappers (snakefiles) [16] were used to automate processes. Snakemake is able to detect missing input files, for which it automatically performs previous rules (processes) that generate the missing input. The pipeline was also made more cohesive by removing hard-coded parameters from the scripts, replacing them with command-line parameters and specifying these in the snakefiles.

Below, we outline the three main steps in the pipeline in more detail. For a more detailed explanation of the pipeline, see 6.1 Supplement 1.

2.4 Data generation

To modularize the generation of the training data, processes were divided into three modules: generation of the ancestral sequence, generation of derived/simulated variants and annotation of generated variants.

2.4.1 Generation of the ancestral sequence

The new CADD implementation makes use of already constructed ancestral sequences that are constructed by Ensembl release 108 [17]. These ancestral sequences are extracted from EMF (Ensembl Multi Format) files, obtained from the Ensembl database. The Multiple Sequence Alignments of these files are split into alignment blocks, each with its own phylogeny depending on the aligned species. For each internal node of the phylogeny, ancestral sequences are provided. Thus, an ancestral sequence for a species is split over multiple alignment blocks. The previous implementation made use of MAF (Mutation Annotation Format) files, which are very similar to EMF files. But since Ensembl changed its standard format for storing genomic alignments to EMF, the new implementation uses these files, converts them to MAF, marks the correct ancestral sequences in each alignment block and processes them using MafTools v3.15 (mafDuplicateFilter, mafStrander, mafRowOrderer, mafSorter) [18]. Python is used to extract the correct ancestral sequences from the alignment blocks and reconstructs the whole ancestral sequence. To allow iterating over multiple MAF files, MafTools processes are wrapped in Python wrappers.

The new pipeline was run for the generation of the mouse-rat ancestral sequence. EMF files of 21 *murinae* species (including mouse GRCm39 and rat mRatBN7.2) were obtained from Ensembl release 108 [17] and the mouse GRCm39 reference genome was obtained from the NCBI database [19].

2.4.2 Generation of derived and simulated variants

Derived variants are generated from the ancestral sequence. Similar to the previous implementation, the new implementation makes use of the ancestral sequence alongside a reference genome (of the species of interest). Reference genomes are often constructed based on a single organism, therefore variant population frequencies (single nucleotide polymorphisms (SNPs) specifically) are required to introduce more fixed variants from different individuals of the same species. Reference unique variants are excluded. The variant population frequencies file is processed with Vcftools v0.1.16 [20], which divides the content of the file and splits its content over new files for each chromosome. Afterwards, derived variants are filtered to retain only SNPs.

The estimation of nucleotide substitution rates for the generation of simulated variants is different than in the previous implementation. Simulated variants are generated based on the reference genome and the estimated nucleotide substitution rates. These rates are determined from the number of different types of nucleotide substitutions (e.g. Adenine to Cytosine, Adenine to Guanine, et cetera.), by iterating over the genome and counting the nucleotide differences between the reference genome and the generated ancestor. The rates in the previous implementation were inefficiently determined from EMF files, since it iterates over nucleotides of sequences split over multiple alignment blocks.

In the previous implementation, the parameter for the number of events (approximate number of substitution events) needed to be manually adjusted after the first run for the generation of the simulated variants, to eventually obtain a similar number of simulated variants as derived variants. In the new implementation, the number of substitution events is automatically set relatively high (to 20,000,000), to generate more simulated variants than derived variants. Afterwards, simulated variants are filtered for variants that are located at positions where an ancestral sequence is defined. This is followed by trimming the number of simulated variants, resulting in equal numbers of

simulated and derived variants. This removes the manual user input previously needed for the generation of similar numbers of simulated and derived variants, at the cost of more computational work.

The new pipeline was run to generate both derived and simulated variants. For the generation of derived variants, the variant population frequencies (Accession: PRJEB53906) were obtained from The Mouse Genome Project 2013-2021 [21].

2.4.3 Annotation of generated variants

After generating the variants, they are annotated with various genomic-derived annotations. The variants are given annotations that are considered basic by Christian Groß: VEP annotations [22], GERP conservation scores [23], PhastCons scores [24], PhyloP scores [24], Grantham scores [25], the genomic position of the variant and GC% / CpG%.

In addition to these basic annotations, one more annotation is added to the features in the new implementation. This annotation indicates whenever the variant is located on a position of a repeat. Most types of repeats are not conserved, therefore non-conserved repeats are expected to be useful for differentiating the two classes of variants, since variants in non-conserved elements are expected to be less deleterious than variants in conserved elements.

After annotation, variants are post-processed: categorical features are one-hot-encoded and missing values are imputed with the mean of the corresponding feature from the simulated variants only.

The new pipeline was run on the generated mouse variants. Grantham scores were obtained from the research paper '*Amino Acid Difference Formula to Help Explain Protein Evolution*' [25]. GERP annotations for GRCm39 were obtained using the Ensembl Perl API v98 with a Perl script [23], which gathers the scores directly from the Ensembl release 108 database. Repeat positions, PhastCons and PhyloP scores were downloaded for GRCm39 from the UCSC genome annotation database 2020 [24].

2.5 Model training & testing

After annotating the generated variants, a classification model is trained and evaluated. Similarly to the previous implementation, the new implementation merges the annotated variants and scales each feature with its standard deviation (standard scaler). The new implementation makes use of a newer version of GraphLab, called Turi Create v6.4.1 [26]. Turi Create is used to split the dataset between a training set (80%) and a validation set (20%) and then train a logistic classifier with L2-norm penalization to prevent overfitting. After training, the model is tested on the validation set and outputs for each variant their posterior likelihood for both classes (deleterious and benign, class 0 and 1 respectively). This is followed by their built-in model evaluator, which evaluates the model based on their performance on the validation set (with its accuracy, precision, recall and AUC).

Due to time constraints, the new implementation was only run on the generated datasets (annotated derived and simulated variants) of mouse chromosome 19.

2.6 Generation of genome-wide CADD scores

After training and evaluating the model, genome-wide CADD scores are generated. Similar to the previous implementation, the new implementation generates a VCF file with all possible variants at every position in the genome. All variants are afterwards annotated and post-processed as described above.

To decrease the required RAM memory, the annotated VCF file is divided into chunks of 1,000,000 (parameter, default value) lines. Afterwards, the features in every chunk are scaled with the previously calculated standard deviation. The posterior likelihoods are derived from the trained model on the chunks, the chunks are merged back into one file and the variants are sorted in descending order on the posterior likelihoods of class 0 (deleterious variant). Afterwards, PHRED-like scores are calculated based on the posterior likelihoods (RAW scores). This is performed by mapping the posterior likelihoods to a PHRED-like scale from 1 to 99 based on their rank relative to the total number of every possible variant in the mouse reference:

$$-10 * \log_{10}\left(\frac{rank}{total\ number\ of\ variants}\right).$$

PHRED-like scores are sorted in ascending order on the genomic position of the variant. At last, summary files are created that contain the minimum, the median, the maximum and the standard deviation of the three CADD scores that occur at any given position.

Due to time constraints, the new implementation was only run for the generation of mouse CADD scores in chromosome 19.

3. Results & Discussion

3.1 Increased pipeline automation efficiency with Snakemake

Snakemake was used to automate the execution of the various processes in the CADD pipeline. In comparison to other alternatives for workflow management methods such as Temporal [27], Snakemake is more user-friendly and easier to use. Currently, the pipeline consists of five snakefiles (Snakemake wrappers used to automate processes). Each snakefile manages processes for its distinctive modules. These processes were modularized to make them easier to read and use. To apply the pipeline to different species, the parameters in the snakefiles need to be adjusted and EMF files, reference genome, variant population frequency file and annotations (Repeat positions, PhastCons and PhyloP scores) needs to be downloaded.

In comparison to the previous implementation, the new implementation only needs one manual step for each module, while the previous implementation needed between 5 to 15 manual steps for each module. This makes the new pipeline more cohesive in comparison to the previous implementation.

Python wrappers were used throughout the pipeline, for command line tools (such as MafTools) and already-written Perl/Python scripts, so they would be able to iterate over multiple input files.

3.2 Performance of the generation of the ancestral sequence

Similar to the previous implementation, the new implementation is dependent on Ensembl's EMF files. These are generated by the EBI Ensembl team and contain constructed ancestral sequences split over multiple alignment blocks. These ancestral sequences are used in the pipeline by default, since the Ensembl database already has large numbers of different ancestral species. The Ensembl EMF files can be directly used in the pipeline without any modifications. However, ancestral sequences from other sources or personally constructed ones (in FASTA format) can also be used in subsequent steps.

To determine proper construction of the mouse-rat ancestral sequence with the new implementation, it was compared to the mouse reference. The chromosomes of the ancestral sequences and the corresponding chromosomes of the mouse reference were determined to be of the same length. Both chromosome 19 sequences had a length of 61,420,004 bp [28], which was expected since the mouse-rat ancestor was constructed with the mouse genome as its reference. IGV (Integrative Genomics Viewer) was used to check for differences between the ancestral sequence and the mouse reference. With the exception of gaps and derived variants, the two sequences were very similar. In chromosome 19 the difference between the ancestral sequence and the mouse reference, was 1,086,198 derived SNPs, roughly 1.77% of the whole chromosome (61,420,004 bp) [28]. The mouse-rat ancestor could be further validated by performing a BLAST between the mouse-rat ancestor and the mouse reference genome, to determine their similarities. But due to time constraints, this was not further investigated.

Furthermore, the ancestral sequence of the Y chromosome does not seem to be constructed, since there were no mouse-rat ancestral sequences marked within the alignment blocks. This could be due to extensive genetic decay, similar to how humans lost 97% of their ancestral genes in their Y chromosomes [29].

3.3 Implementation changes for the generation of the variants

The new implementation generates derived variants by determining nucleotide differences between the ancestral sequence, the mouse reference and the variant population frequencies (SNPs). For the simulated variants, the nucleotide substitution rates are determined from the number of different types of nucleotide substitutions (Adenine to Cytosine, Adenine to Guanine, et cetera.) between the mouse reference and the generated ancestor, while it was previously extracted from the EMF files. This change is made to make the extraction of the nucleotide substitution rates more efficient, since it is simpler to determine the different nucleotide substitutions when you would iterate over the nucleotides of the two sequences, than to iterate over the nucleotides of sequences split over multiple alignment blocks. It still uses the sequence composition of the derived variants (the substitution rates) to simulate a matching set of *de novo* simulated variants [13].

In the previous implementation, the generation of the simulated variants needed quite some manual input. The parameter for the number of events (approximate number of substitution events) needed to be adjusted for the generation of the simulated variants, to obtain similar numbers of simulated and derived variants. However, due to the number of events not directly correlating with the number of simulated variants (16.639.795 variants were simulated with 20.000.000 events), at least two runs (script executions) were needed. The first run can be performed with any number of events, for which the number of simulated variants needed to be checked after simulation. If the number of simulated variants did not result in roughly the same number of the derived variants, in the subsequent runs the parameter for the number of events needed to be increased or decreased until the number of simulated and derived variants were roughly the same. To remove this manual step from the pipeline, the number of events is by default set relatively high in the new implementation. Afterwards, the large number of simulated variants is trimmed (by removing simulated variants) to eventually obtain the same number of simulated and derived variants.

In total, 44,011,391 derived SNPs are determined across the whole mouse genome. The length of mouse genome GRCm39 is 2,728,222,451 bp [28], which means that roughly 1.6% of the whole mouse genome is determined to consist of derived SNPs. The previous (mouse) mCADD paper resulted in 33,622,843 derived SNPs across the whole genome [2] (GRCm38/mm10; 2,730,855,475 bp [30]), which makes 1.2% of their mouse genome to consists of derived SNPs. This is quite similar to the results in the new implementation. However, there is a difference of 24% between the two numbers of derived SNPs. This difference is most likely caused, due to the use of two different versions of the reference genome (GRCm39: scaffolds=102, scaffoldN50=106,145,001 [28]; GRCm38/mm10: scaffolds=162, scaffoldN50= 54,517,951 [30]) and the use of two different versions of the rat genome (mRatBN7.2 and Rnor_5.0 for the mouse-rat ancestor).

To determine proper generation of the simulated variants, the nucleotide substitution rates used for the simulation were compared to the substitution rates extracted from the simulated variants after its simulation. This comparison resulted in very similar substitution rates between the two (Figure 3). Furthermore, the transversion and transition rates were determined to be 36% and 64% respectively in the simulated variants, which matches roughly with the natural transversion and transition rates of 34% and 66% (1:2) [31] (roughly the same in *murine* species [32]).

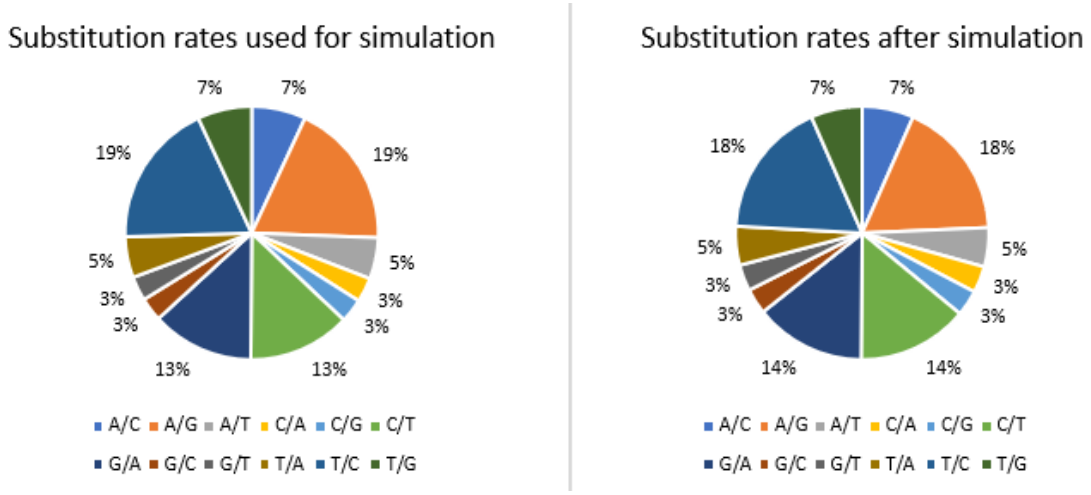

Figure 3: Substitution rates used to simulate variants (left) and determined from the resulting simulated variants (right).

3.4 Addition of new annotations/features

In the new implementation, features can be added by adjusting the snakefile, which is not that different compared to the previous implementation. To add new features, it needs to contain information about the genomic positions of the feature. Afterwards, the features are appended to the previously annotated variants based on their genomic position.

In the new implementation, a new feature is used. This new feature indicates whether the variant is located on the position of a repeat with binary numbers (0/1). While the model was trained, weights were assigned to the features. The more positive the weight of a feature is the more it contributes to the classification of class 1 (benign variant), while the more negative it is the more it contributes to the classification of class 0 (deleterious variant). The repeat position feature was assigned a weight of 0.051, this means that it has some contribution to the prediction of class 1. However, this feature seems to be negligible, since the model performance did not change after the removal of the feature.

Currently, the pipeline uses the positions of every repeat in GRCm39 from the UCSC database [24]. Annotations from the UCSC database are used, since UCSC has a wide range of genomic annotations for lots of different species. Alternatively, positions of only specific repeats (such as only conserved repeats like Tandem Repeats (TR) [33] or Long Terminal Repeats (LTR) [34]) could be used as features instead, as these might be more useful for differentiating the two variants.

3.5 Model evaluation

The latest iteration of hCADD [13] makes use of Scikit-learn, while the previous livestock implementation (chCADD) uses GraphLab. Compared to Scikit-learn, Graphlab (or Turi Create) is very scalable, therefore it performs well with large datasets.

After training (with the generated training set of chromosome 19), the model was evaluated with the validation set (Table 1) and the weights of the features were checked (Figure). The accuracy indicates how often the model predicts the class correctly, the precision indicates how many predictions for a specific class are predicted correctly, the recall indicates how many of the items (variants) that belong to the same specific class are predicted correctly and the AUC (Area Under the ROC Curve) is a measure of separability that indicates how well the model is able to distinguish the two classes. These scores range from 0 to 1, the closer the scores are to 1, the better the model.

Confusion matrix		
Target label	**Predicted label**	**Count**
0	1	7748
0	0	125136
1	1	110881
1	0	21065

Table 1: Resulting confusion matrix after the evaluation of the trained model. The trained model was trained and evaluated on the generated dataset of chromosome 19. Benign (derived) variant: label 1; Deleterious (simulated) variant: label 0.

The model of the new implementation results in an accuracy of 0.891, a precision of 0.934, a recall of 0.840 and an AUC of 0.946. With top features shown in Figure B. In comparison to the mCADD paper by Christian Groß [2], their models were only evaluated with the receiver operating characteristic (ROC-AUC), which resulted in AUC scores of roughly 0.666. The top-performing features consist of mostly position and conservation-based features. This is unexpected, since the genomic positions are normally not correlated with the deleteriousness of an SNP. Compared to the mCADD paper by Christian Groß [2], only GC%, GERP scores and SIFT scores are shared between top-performing features from the previous and the new implementation.

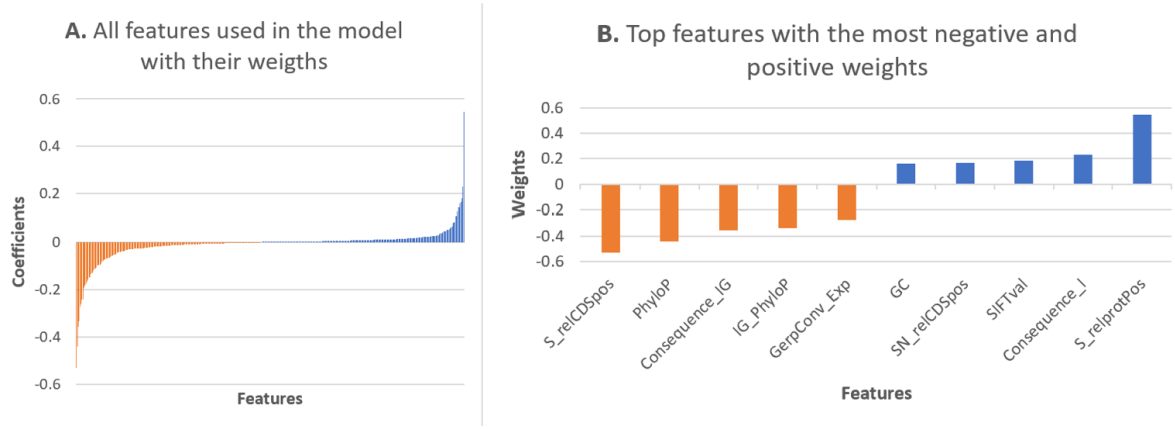

Figure 5: Two bar graphs that show the weights (coefficients) of the used features (orange=negative weight, blue=positive weight). The y-axis represents the value for the weights and the x-axis represents the features. Graph A shows the weight of all features, while graph B shows the top ten features with the most negative and positive weights. The five features with the most negative weights are Relative coding sequence position of a splice donor variant, PhyloP score, Intergenic variants, PhyloP scores of intergenic variants and GERP conservation scores. The five features with the most positive weights are Protein position that is also in a splice site, Intronic variants, SIFT score, Relative coding sequence position of synonymous variants and GC% calculated based on a 75 nucleotide window.

3.5.1 Noticeable high weight for genomic position-based features

The feature for the genomic position was not used previously in Christian's mCADD and chCADD models. The current feature for the genomic position is unscaled, which resulted in a weight of 2.78e-7. However, when scaled like the rest of the features, it results in an unusually high positive weight (4.836) with similar model performance.

As to why the weight of position-based features is higher than usual, is due to a mistake in the trimming of the simulated variants. Simulated variants were first generated over whole chromosomes and were trimmed afterwards at the tail of the dataset, this resulted in an unintended sampling of the simulated variants on their genomic positions (while the derived dataset contains variants located over whole chromosomes). To prevent this problem, trimming should happen at random.

After solving the sampling problem, the top features are checked (Figure). Conservation-based features are among the top-performing features, similar to Christian's mCADD model [2]. PhastCons scores, GERP scores and SIFT scores are shared between top-performing features from the previous and the new implementation. Weights of position-based features seem to have dropped towards 0. The genomic position got a weight of 1.34E-09. This weight became much closer to 0, thus it has very little predictive value for the prediction of the variants, which is expected.

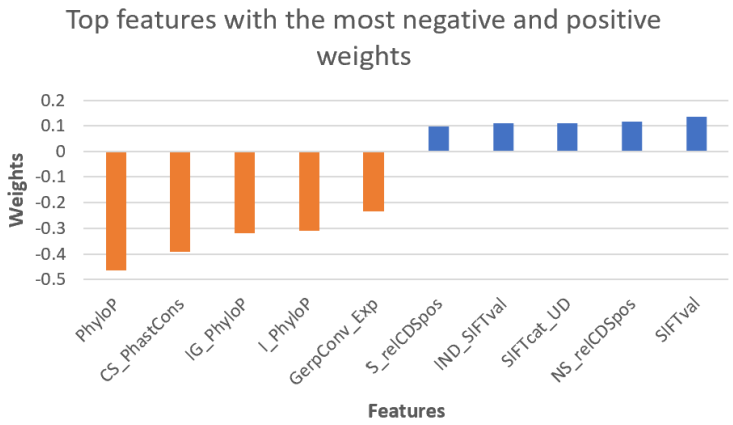

Figure 5: The top ten features with the most negative and positive weights after solving the sampling problem. The five features with the most negative weights are PhyloP scores, PhastCons scores in canonical splice sites, PhyloP scores in intergenic variants, PhyloP scores in intronic variants and GERP conservation scores. The five features with the most positive weights are Relative coding sequence position of a splice donor variant, SIFT score indicator, SIFT category of change, Relative coding sequence position of non-synonymous variants and SIFT score.

3.5.2 Noticeable high-model performance

Furthermore, the overall model performance of the new implementation is much higher in comparison to the previous mCADD models (AUC of roughly 0.666). This is unexpected, since the features in the new implementation were also used in the previous mCADD model (except for repeat position, which was negligible). Therefore similar performance is expected. Even without the feature for the genomic position, the model still performed better in comparison to the performance of the previous mCADD model (Table 2.3).

The cause for the higher performance is suspected to be influenced by the sampling problem, since it may have effects on other features. To check whether this is the case, the model is retrained with the fixed simulated dataset (Table 2**Error! Reference source not found.**). From these results, it seems that the model still performs better in comparison to Christian's mCADD model even with the sampling problem solved.

	Accuracy	Precision	Recall	AUC
1. Christian's mCADD models	/	/	/	~0.666
2. With genomic position	0.891	0.934	0.840	0.946
3. Without genomic position	0.746	0.739	0.760	0.809
4. With genomic position (And solved sampling problem)	0.721	0.658	0.509	0.785
5. Without genomic position (And solved sampling problem)	0.721	0.664	0.507	0.785

Table 2: Performance of different models with or without the feature for genomic position and the solved sampling problem.

Another possibility for the high model performance is the use of newly updated annotations, since newer annotations are assumed to be more accurate and therefore possibly better for the model to distinguish the two variants. For example, the use of differently computed PhyloP and PhastCons scores, since the previous scores in the Christian's mCADD models were computed without the mouse reference taken into account. These were based on a 60-taxa Vertebrate alignment, a 40-taxa Placental alignment, a 21-taxa Euarchontoglire alignment and a 8-taxa Glire alignment (all excluding the mouse) [2], while the PhyloP and PhastCons scores in the new implementation were computed by UCSC [24] from 35 Vertebrate alignments including mouse genome GRCm39. This may explain the better performance, since a more updated feature (or even a more species-specific feature) is able to improve the performance of the model to differentiate the two classes of variants.

3.6 Performance on the generation of CADD scores

To reduce the required computational time to complete the task, this module was originally run manually in parallel. The new implementation uses Unix parallel commands in the snakefile, but may not be as effective when used with computer clusters that use SLURM, since it would need a high amount of RAM memory for a single job. This would cause the SLURM queue to be longer. Alternatively, parallelization could be downscaled to reduce the required RAM (by running fewer chunks in parallel). Jobs can also be manually (or automatically) submitted to SLURM, instead of using the Unix parallel commands in the snakefile.

After solving the sampling problem, the generation of CADD scores is reperformed for mouse chromosome 19. Both high as well as low CADD scores are generated. At roughly around the 10,004,070th position there are variants with unusually high CADD scores (above 50) that may belong to functional important genes. Few of these variants are located on the gene Rab3il1, which is involved in exocytosis and is expressed in embryos [35].

To further investigate the CADD scores, these could be loaded into a genome browser alongside the coding sequences in mouse chromosome 19. It is expected that high CADD scores are located on these coding sequences, but due to time constraints, this was not further investigated.

4. Conclusions & Recommendations

4.1 Conclusions

Overall, the new implementation of the CADD pipeline works, as genome-wide CADD scores were eventually generated for mouse chromosome 19 (GRCm39). However, due to the sampling problem with the simulated variants, it resulted in questionable weights for position-based features. This was eventually solved, the model was retrained and CADD scores were regenerated for chromosome 19. The CADD scores and the model performance could be further investigated to improve the pipeline.

Furthermore, the new implementation is more cohesive, due to the automation of its processes with Snakemake and the pipeline is also simpler to use for different species. Even though the new implementation performs well with high model performance, it has still some flaws that could be improved upon.

4.2 Recommended improvements

4.2.1 Further output validation

To further validate the mouse-rat ancestor, a BLAST between the mouse-rat ancestor and the mouse reference genome could be performed, to determine their similarities. It is expected that the two sequences will be very similar. Furthermore, the CADD scores could be validated by loading the scores into a genome browser alongside the coding sequence. It is expected that high CADD scores are located on these coding sequences.

4.2.2 Further performance validation

It is recommended to further investigate the performance of the pipeline. The trained model could be tested on new validation sets, to assess the model performance in predicting SNPs from a validation set where the phenotype-altering effects of each SNP are known with certainty. Such as Fairfield, Mutagenetix or ClinVar-ESP datasets, which were also used in the mCADD paper by Christian Groß [2].

The new CADD implementation should theoretically work for different species. To determine whether it could be run for a different species, the new implementation should be performed to generate e.g. sheep or other livestock CADD scores. To generate sheep CADD scores (or CADD scores for another species), EMF files, the reference genome, variant population frequencies, GERP annotations, Repeat positions, PhastCons and PhyloP scores are needed. EMF files and GERP annotations are obtained from the Ensembl database. The reference genome can be obtained from NCBI. Variant population frequencies are obtained from the International Sheep Genomics Consortium [36] [37]. Repeat positions, PhastCons and PhyloP scores are obtained from the UCSC genome annotation database.

4.2.3 Pipeline improvements

Additionally, there is still room for improvement in the code. The new implementation consists of multiple snakefiles for the automation of its processes. Alternatively, a main snakefile could be used to connect every snakefile together to make the pipeline even more cohesive. This would make it easier to use, since only one snakefile is needed to run the whole pipeline.

The new implementation constructs the ancestral sequence also quite inefficiently. The EMF files are converted to MAF format followed by a few processing steps to eventually extract the correct ancestral sequence from the processed MAF files. Instead, the ancestral sequence should directly be extracted from the EMF files and the subsequent processing steps could also be combined into a

single script to reduce the number of scripts used in the pipeline. This will make the pipeline overall clearer for users.

Furthermore, the accessibility of the pipeline could be improved for users with limited bioinformatics knowledge. An option to improve the accessibility of the pipeline, is to run the snakefiles through a web page with adjustable parameters, that sends Unix commands to the server. Alternatively, the pipeline could also be converted to a galaxy workflow with Galaxy Tool Factory [38] and be run on a (private) Galaxy server [39]. Galaxy is an easy-to-use browser-based workbench for scientific computing and with Galaxy Tool Factory all different kinds of scripts (e.g. Python, Perl, R or bash) can be run through the Galaxy server.

5. Data availability

The developed pipeline is available at the Wageningen University & Research (WUR) GitLab page (only accessible with a WUR account): 'https://git.wur.nl/seyan.hu/cadd_pipeline'

For the data generated throughout the development of the pipeline, check the following path on the Lustre filesystem: '*/lustre/nobackup/WUR/ABGC/hu052/*'.

5.1 Contact information:

- Thesis student:
  - Seyan Hu, 1128566
  - WUR mail: *seyan.hu@wur.nl*
  - Personal mail: *seyanhu@hotmail.com*
- Main-supervisor from Animal Breeding & Genomics Group:
  - Martijn Derks
  - WUR mail: *martijn.derks@wur.nl*
- Co-supervisor from Bioinformatics Group:
  - Dick de Ridder
  - WUR mail: *dick.deridder@wur.nl*

The pipeline was developed with Python v3.6.13 in different Conda environments, due to conflicting packages caused by the Ensembl Perl API. Due to this error, two environments were used, one with the Ensembl Perl API that was solely used in obtaining GERP scores and one environment for the other processes. This could make the pipeline more difficult to use. However, it can be solved by manually downloading the GERP scores from the Ensembl database with Unix commands (similar to how EMF files, Repeat positions, PhastCons and PhyloP scores were previously downloaded) instead of using the Ensembl Perl API.

Backups of the different environments are available on the WUR GitLab page.

6. Appendix

6.1 Supplement 1: Detailed Materials & Methods

6.1.1 Generation of the ancestral sequence

The snakefile for the generation of the ancestral sequence performed the following:

The EMF files were reformatted to MAF files with Ensembl's *'emftomaf_bash.sh'* Perl5 v26 script from the Ensembl-compara Github page. The correct ancestral sequences (mouse-rat ancestor) were marked in each alignment block by the script *'marking_ancestor.py'*. Afterwards, MafTools v3.15 [18] was used: to remove duplicate sequences in the alignment blocks (mafDuplicateFilter); to flip alignment blocks in which the mouse and its mouse-rat ancestor were on the negative strand (mafStrander); to retain only sequences in the alignment blocks from the mouse, rat and mouse-rat ancestor (mafRowOrderer); and to reorder the sequences in the alignment block, so that the mouse will be the first species in any alignment block (mafRowOrderer). These MafTools processes were performed with the Python wrappers *'maftools_duplifilter.py'*, *'maftools_strander.py'*, *'maftools_roworderer.py'* respectively. Since alignment blocks from the same chromosome were divided between multiple MAF files, *'chr_sorting.py'* was performed to merge MAF files corresponding to the same chromosome. This will result in one MAF file per chromosome. MafTools's mafSorter was performed with the Python wrapper *'maftools_sorter.py'* to sort alignment blocks with respect to the coordinates of the mouse genome (GRCm39). At last, unwanted mouse strains (any strains other than GRCm39) were removed from the alignment blocks with *'rm_species.py'*. Following this process, the marked ancestral sequences from the processed MAF files were extracted and reconstructed with the GRCm39 reference by the script *'wrapper_gen_anc_seq.py'* (which acts as a Python wrapper for *'gen_ancestor_seq_v2.py'*).

6.1.2 Generation of the derived and simulated variants

The snakefile for the generation of derived and simulated variants performed the following:

The variants population frequency file was split into individual VCF files per chromosome with the Python wrapper *'gen_freq.py'* for Vcftools v0.1.16 [20]. Afterwards, the derived variants were generated per chromosome with '*wrapper_derived_gen.py'* (which acts as a Python wrapper for *'derived_var_gen.py'*). This script will iterate over the nucleotides of the previously reconstructed ancestral sequence and compares it to the corresponding mouse GRCm39 chromosome alongside the previously split population frequency files. The derived variants were then filtered with '*filter_derived_for_snps.py*' for SNPs.

Afterwards, nucleotide substitution rates were extracted per chromosome with '*wrapper_substitution_calc.py*' and '*nt_substitution_calc.py'*. Following this, '*fix_chr_fai.py'* was performed to replace the NCBI chromosome identifiers with the correct chromosome number for each index file (FAI file) of GRCm39. With the (1.) extracted substitution rates, (2.) the corresponding index file and (3.) the number of events, the simulated variants were generated per chromosome by '*wrapper_simulated_gen.py'* (which acts as a Python wrapper for '*simulated_var_gen.py'*). For the generation of simulated variants, the number of events was automatically set relatively high. Following this process, the simulated variants were filtered with '*filtering_simulated_for_ancestor_site.py'.* This removes variants that were simulated on positions that correspond to a gap in the ancestral sequence. Afterwards, the number of simulated variants was trimmed with '*trim_simulated_varV2.py'*. The number of simulations was automatically adjusted

by setting the parameter for events to a relatively high number (set to 20,000,000), so that after trimming the number of simulated variants will result in the same number as derived variants.

6.1.3 Annotation of the generated variants

To perform the VEP annotation, the VEP library for GRCm39 was installed with the command line '*vep_install -a cf -s mus_musculus -y GRCm39 -c ./.vep --CONVERT*'. The snakefile for the annotation of the generated variants performed the following:

VEP annotation was performed in offline mode for the generated variants by the Python wrapper '*vep_wrapper.py'* with the installed GRCm39 VEP library. The VEP annotated variants were compressed and indexed with bgzip v1.9 and tabix v1.9 respectively. Afterwards, the '*Extra*' column of the VEP annotated variants were processed with '*vep_output_processing_wrapper.py'* (which acts as a Python wrapper for '*VEP-processing.py*'). Concurrently, corresponding Grantham scores and GC% / CpG% were added to the annotations, resulting in a tab-delimited file.

The Python wrapper '*Ensembl_API_Gerp_query.py'* was performed to obtain per chromosome the GERP conservation scores and GERP element conservation scores using the Ensembl API scripts ('*Ensembl_API_Gerp_conservation_query.py'* and '*Ensembl_API_Gerp_element_query.py'* respectively). Afterwards, the GERP element conservation scores were reformatted in a similar format to the GERP conservation scores with '*format_GerpElem.py'*. This made it easier to search for scores at positions of the generated variants.

The two downloaded PhastCons and PhyloP files (in BigWig format) were converted to Wig format with UCSC's bigWig2Wig v377 tool, to make the BigWig files human readable. With '*format_pC_pP_wig.py'* the two Wig files were reformatted similarly to the GERP conservation scores. Afterwards, the two reformatted files were split with '*split_pC_pP_scores.py'*, resulting in one PhastCons file and one PhyloP file per chromosome.

For the repeats, '*get_position_repeats.py'* was performed to extract per chromosome the positions of the repeats from the FASTA files (obtained from the UCSC database). Since repeats are soft-masked with lower cases, the positions can easily be extracted. In the end, the script outputs a list of the positions of the repeats.

Finally, the gathered annotations were merged per chromosome into a tab-delimited file by appending the correct annotations to the VEP annotated variants, with '*merge_all_annotationsV3.py'*. This resulted in fully annotated derived and simulated variants. Afterwards, '*encoding_mv_wrapper.py'* was performed to handle categorical features with one-hot encoding and missing values with imputation. Imputation was performed with the mean of the corresponding feature from the simulated variants only. Per chromosome, this resulted in two sets of completely annotated variants, one derived set and one simulated set. The mean used for the imputation was also saved, since this will later be used for the generation of genome-wide mCADD scores.

6.1.4 Model training & evaluation

The snakefile for the training and evaluation of the model performed the following:

The snakefile loads the datasets as DataFrames and merges them afterwards with the script '*feature_scaling.py*'. This was followed by the scaling of the features with their standard deviation

(standard scaler). Eventually, it outputs a CSV file and a dictionary file that contains the features as the key and its standard deviation (that was used for scaling) as the value. This dictionary was later used in the generation of genome-wide mCADD scores for the scaling of its features.

This was then followed by the script, '*train_test_model.py'*. This script iterates over the scaled DataFrames and splits each DataFrame into a training (80%) and validation (20%) set. Data in the training set was fitted to the parameters of the logistic classifier, with the L2 norm penalization. Afterwards, the fitted model was evaluated on the validation set by the built-in evaluator from Turi Create. Per chromosome, the script saved the fitted model and outputs a file containing the evaluation results and a file containing the weights of the used features.

6.1.5 Generation of genome-wide CADD scores

The snakefile for the generation of genome-wide CADD scores performed the following:

First, it generated per chromosome at every position in the reference genome all possible variants, with '*gen_all_variants.py*'. Per chromosome, it resulted in a VCF file that contains at any given position the three possible variants.

VEP annotation was performed on these variants in offline mode by the Python wrapper '*vep_wrapper.py'* with the previously installed GRCm39 VEP library. Similar to the annotation of the training set, the VEP annotated variants were processed with '*vep_output_processing_wrapper.py*' (which acts as a Python wrapper for '*VEP-processing.py*'). Afterwards, 'merge_all_annotationsV3.py' was performed to add for every variant the correct Gerp scores, PhastCons scores, PhyloP scores and repeat position.

The fully annotated variants were then divided into chunks of 1,000,000 lines. In parallel, the missing values and categorical features were imputed and encoded by '*encoding_mv_wrapper.py'*. Imputation was performed with the previously saved means. Afterwards, the features in each chunk were scaled in parallel with the previously saved standard deviations.

'*ML_predict.py*' iterates over the chunks in parallel and loads the corresponding trained model. The trained model was used to predict the posterior likelihood of the variants for class 0 (deleterious variant) and class 1 (benign variant). The script outputs per chunk a file containing the posterior likelihood for class 0 and class 1 respectively, the position of the variant in the chromosome, the reference nucleotide and the alternative nucleotide. These outputs were processed with '*process_prediction.py*', by removing the posterior likelihood for class 1 (benign variants), since only the probability of class 0 (deleterious variants) was needed.

The chunks were merged per chromosome into one file, with 'merge_chunks.py'. The variants in the merged chunks were sorted with '*sort_proba.py'* in descending order on the posterior likelihoods of class 0. At last, PHRED-like scores were assigned to the posterior likelihood by 'assigning-PHRED-scores.py' and the scores were resorted on the genomic positions of the variants. Summary files were created that contain the minimum, the median, the maximum and the standard deviation of the three CADD scores that occur at any given position with '*compute_CADD_summary.py'*.

6.2 Supplement 2: Data management plan

Data management plan belonging to the MSc thesis performed at the Animal Breeding and Genomics Group by Seyan Hu, completed in March 2023.

Agreements

1. The data used in this thesis project have been described in this document and have been stored in a systematic manner (at least in separate folders for all sections as described below). Data includes all data as mentioned in the results section of your report.
2. The data management plan has been discussed with the MSc thesis supervisor and he/she has agreed on the location for data storage.
3. In case of confidentiality, contact details of the responsible person from the company/institution that has ownership of the data are mentioned in this document.
4. **The data can be found in/on/through the Lustre filesystem ('*/lustre/nobackup/WUR/ABGC/hu052/*') / the Wageningen University & Research (WUR) GitLab page (https://git.wur.nl/seyan.hu/cadd_pipeline) / Martijn Derks.**

6.2.1 Section A - Raw data

Files	Received from	On date
EMF files of 21 *murinae* species	Ensembl release 108	9-2022
GRCm39 reference	NCBI database	9-2022
Mouse variant population frequencies (Accession: PRJEB53906)	The Mouse Genome Project 2013-2021	10-2022
Grantham scores	The research paper '*Amino Acid Difference Formula to Help Explain Protein Evolution*'	11-2022
GERP annotations for GRCm39	Ensembl release 108	11-2022
Repeat positions for GRCm39	UCSC genome annotation database 2020	11-2022
PhastCons scores for GRCm39	UCSC genome annotation database 2020	11-2022
PhyloP scores for GRCm39	UCSC genome annotation database 2020	11-2022

6.2.2 Section B - Data analysis and scripts

File names	Created in (month, year)	Remarks
Snakefile_gen_ancestral_V2	9-2022	
emftomaf_bash.sh	9-2022	Converts emf files to MAF format.
marking_ancestor.py	9-2022	Identifies the last common ancestor between two given species and marks it with an identifier.
maftools_duplifilter.py	9-2022	Removes all duplicate sequences and keeps only the one sequence that is the most similar to the block consensus.
maftools_strander.py	9-2022	Flips all alignment blocks in which the species of interest and its ancestors have been on the negative strand.
maftools_roworderer.py	9-2022	Reorders species within any alignment block, so that the wanted species are in front.

chr_sorting.py	9-2022	Creates one MAF file for each chromosome.
maftools_sorter.py	9-2022	Sorts alignment blocks with respect to coordinates of the first species of interest using its genome.
rm_species.py	9-2022	Removes other species with similar names to the species of interest from all alignment blocks.
rm_opposite_strand.py	9-2022	Will remove alignment block with sequences that are still on the minus strand.
wrapper_gen_anc_seq.py	9-2022	Reconstructs the marked ancestor sequences in the preprocessed MAF files using the identifiers and outputs per chromosome a FASTA file of the ancestral sequence.
Snakefile_gen_variants	10-2022	
gen_freq.py	10-2022	Generates frequency files form the population variants (VCF files).
wrapper_derived_gen.py	10-2022	Generates the derived variants by looking at all data sources (ancestral sequence, reference genome, population frequency files) simultaneously.
filter_derived_for_snps.py	10-2022	Filters the derived variants for SNPs and indels.
wrapper_substitution_calc.py	10-2022	Obtains parameters for the generations of simulated variants.
fix_chr_fai.py	10-2022	It happens that the chr in the index file is given a uncommon name (a NCBI identifier), this script will fix it.
wrapper_simulated_gen.py	10-2022	Generates the simulated variants using the nucleotides substitution rates and the chromosomes of the genome of the desired species (reference).
filtering_simulated_for_ancestor_site.py	10-2022	Filters the simulated variants for variants that are generated on the ancestral sequence (and not on gaps).
trim_simulated_var.py	10-2022	Trims lines in simulated VCF file to match with the number of lines in the derived VCF files.
Snakefile_gen_annotations	11-2022	
vep_wrapper.py	11-2022	Performs VEP on the derived and simulated variants.

vep_output_processing_wrapper.py	11-2022	Processes the VEP annotation file and returns a tab delimited, encoded file with additional basic annotations.
Ensembl_API_Gerp_element_query.py	11-2022	The script calls up the gerpelem.pl and getgerp.pl and prints out all Gerp scores for the requested chromosome.
format_GerpElem.py	11-2022	Formats the GerpElem file so that it is easier to merge without the use of dictionaries.
format_pC_pP_wig.py	12-2022	The script reformats these scores so that it is easier to parse to the processed VEP output.
split_pC_pP_scores.py	12-2022	Splits the reformatted PhastCons & PhyloP scores per chromosome, so it would be easier to load when merging.
get_position_repeats.py	12-2022	The script creates per chromosome a output file containing a list of the position of its repeats.
merge_all_annotationsV3.py	12-2022	Adds the y (label) to the processed VEP files and adds for every variant the correct Gerp, PhastCons, PhyloP scores and repeat position.
encoding_mv_wrapper.py	12-2022	It handles missing values with imputation (based on the mean in simulated variants)and categorical features with one-hot encoding.
Snakefile_train_test_model	1-2023	
feature_scaling.py	1-2023	It merges the two DataFrames per chromosome and afterwards it scales the features.
train_test_model.py	1-2023	Iterates over the scaled DataFrames. For each DataFrame it splits the data into a training and testing set. The data is then fitted to the parameters, followed by an evaluation of the model on the test set.
Snakefile_genome-wide_CADD	1-2023	
gen_all_variants.py	1-2023	This script generates all possible variants at every position for the FASTA files in the given path.
vep_wrapper.py	1-2023	Performs VEP on the all possible variants.

vep_output_processing_wrapper.py	2-2023	Processes the VEP annotation file and returns a tab delimited, encoded file with additional basic annotations.
merge_all_annotationsV3.py	2-2023	Adds for every variant the correct Gerp, PhastCons, PhyloP scores and repeat position.
chunk_DFv2.py	2-2023	Splits the merged DataFrame into chunks of default=1,000,000 lines.
encoding_mv_wrapper.py	2-2023	It handles missing values with imputation (based on the mean in simulated variants) and categorical features with one-hot encoding.
feature_scaling.py	2-2023	The features are scaled with the standard deviation used previously for the scaling of the derived and simulated data.
ML_predict.py	2-2023	The previously trained model is loaded and used to predict the posterior likelihood of the variants being derived / simulated.
process_prediction.py	2-2023	The script iterates over the chunks and processes the Dataframe by removing the probability for class 1 (benign).
merge_chunks.py	2-2023	The script checks for the available chromosomes in the given path. Iterates over the chromosome numbers and merger per chromosome their chunks.
sort_proba.py	2-2023	The script iterates over the merged chunks and sorts (in descending order) the RAW-scores.
assigning-PHRED-scores.py	2-2023	Takes the sorted (in descending order) RAW-scores and assigning to them a PHRED-like score.
compute_CADD_summary.py	2-2023	This script takes the fully computed and location sorted PHRED scores file and computes per site summary statistics.

6.2.3 Section C - Final data

Files	**Created in (month, year)**	**Remarks**
Generated ancestral sequences	9-2022	In the path: *'/lustre/nobackup/WUR/ABGC/hu052/generate_ancestral_seq/output/dir_generated_ancestor_seq/'*

Generated variants	10-2022	In the path: *'/lustre/nobackup/WUR/ABGC/hu052/generate_variants/output/dir_filter_derived_snp'* and *'/lustre/nobackup/WUR/ABGC/hu052/generate_variants/output/dir_trimmed_simulated_variants/'* and (without the sampling problem) *'/lustre/nobackup/WUR/ABGC/hu052/generate_variants/retrimming/new/'*
Annotated variants	12-2022	In the path: *'/lustre/nobackup/WUR/ABGC/hu052/generate_annotations/output/dir_complete_dataset/'* and (without the sampling problem) *'/lustre/nobackup/WUR/ABGC/hu052/generate_annotations/retrimming/encoding_imputation/'*
Trained model	1-2023	In the path: *'/lustre/nobackup/WUR/ABGC/hu052/train_test_model/output/dir_trained_tested/'* and (without the sampling problem) *'/lustre/nobackup/WUR/ABGC/hu052/train_test_model/retrimming/'*
Generated CADD scores (without the sampling problem)	3-2023	In the path: *'/lustre/nobackup/WUR/ABGC/hu052/generate_genome-wide_CADD/output/dir_PHRED_assigned'* and *'/lustre/nobackup/WUR/ABGC/hu052/generate_genome-wide_CADD/output/dir_PHRED_summary'*

7. References

[1] M. F. L. Derks *et al.*, "Accelerated discovery of functional genomic variation in pigs," *Genomics*, vol. 113, no. 4, pp. 2229–2239, Jul. 2021, doi: 10.1016/J.YGENO.2021.05.017.

[2] C. Groß, D. de Ridder, and M. Reinders, "Predicting variant deleteriousness in non-human species: applying the CADD approach in mouse," *BMC Bioinformatics*, vol. 19, no. 1, Oct. 2018, doi: 10.1186/S12859-018-2337-5.

[3] Q. Chen, C. Dai, Q. Zhang, J. Du, and W. Li, "[Evaluation of performance of five bioinformatics software for the prediction of missense mutations].," *Zhonghua Yi Xue Yi Chuan Xue Za Zhi*, vol. 33, no. 5, pp. 625–628, Oct. 2016, doi: 10.3760/CMA.J.ISSN.1003-9406.2016.05.009.

[4] C. Gross *et al.*, "Evolutionarily conserved non-protein-coding regions in the chicken genome harbor functionally important variation," *bioRxiv*, p. 2020.03.27.012005, 2020, doi: 10.1101/2020.03.27.012005.

[5] M. Kircher, D. M. Witten, P. Jain, B. J. O'roak, G. M. Cooper, and J. Shendure, "A general framework for estimating the relative pathogenicity of human genetic variants," *Nat. Genet.*, vol. 46, no. 3, p. 310, 2014, doi: 10.1038/NG.2892.

[6] N. M. Ioannidis *et al.*, "REVEL: An Ensemble Method for Predicting the Pathogenicity of Rare Missense Variants," *Am. J. Hum. Genet.*, vol. 99, no. 4, pp. 877–885, Oct. 2016, doi: 10.1016/J.AJHG.2016.08.016.

[7] P. Rentzsch, D. Witten, G. M. Cooper, J. Shendure, and M. Kircher, "CADD: predicting the deleteriousness of variants throughout the human genome," *Nucleic Acids Res.*, vol. 47, no. D1, pp. D886–D894, Jan. 2019, doi: 10.1093/NAR/GKY1016.

[8] D. R. Kelley, J. Snoek, and J. L. Rinn, "Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks," *Genome Res.*, vol. 26, no. 7, pp. 990–999, Jul. 2016, doi: 10.1101/GR.200535.115.

[9] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning–based sequence model," *Nat. Methods 2015 1210*, vol. 12, no. 10, pp. 931–934, Aug. 2015, doi: 10.1038/nmeth.3547.

[10] P. Rentzsch, D. Witten, G. M. Cooper, J. Shendure, and M. Kircher, "CADD: predicting the deleteriousness of variants throughout the human genome," *Nucleic Acids Res.*, vol. 47, no. Database issue, p. D886, Jan. 2019, doi: 10.1093/NAR/GKY1016.

[11] C. Groß *et al.*, "pCADD: SNV prioritisation in Sus scrofa," *Genet. Sel. Evol.*, vol. 52, no. 1, p. 4, Feb. 2020, doi: 10.1186/S12711-020-0528-9.

[12] C. Groß *et al.*, "Prioritizing sequence variants in conserved non-coding elements in the chicken genome using chCADD," *PLoS Genet.*, vol. 16, no. 9, Sep. 2020, doi: 10.1371/JOURNAL.PGEN.1009027.

[13] P. Rentzsch, M. Schubach, J. Shendure, and M. Kircher, "CADD-Splice—improving genome-wide variant effect prediction using deep learning-derived splice scores," *Genome Med.*, vol. 13, no. 1, Dec. 2021, doi: 10.1186/S13073-021-00835-9.

[14] Anaconda, "Anaconda Documentation — Anaconda documentation," 2022. https://docs.anaconda.com/ (accessed Dec. 22, 2022).

[15] "The Python Language Reference — Python 3.10.7 documentation." https://docs.python.org/3/reference/index.html (accessed Sep. 14, 2022).

[16] J. Köster and S. Rahmann, “Snakemake—a scalable bioinformatics workflow engine,” *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, Oct. 2012, doi: 10.1093/BIOINFORMATICS/BTS480.

[17] F. Cunningham *et al.*, “Ensembl 2022,” *Nucleic Acids Res.*, vol. 50, no. D1, pp. D988–D995, Jan. 2022, doi: 10.1093/NAR/GKAB1049.

[18] A. Mayakonda, D. C. Lin, Y. Assenov, C. Plass, and H. P. Koeffler, “Maftools: Efficient and comprehensive analysis of somatic variants in cancer,” *Genome Res.*, vol. 28, no. 11, pp. 1747–1756, Nov. 2018, doi: 10.1101/GR.239244.118/-/DC1.

[19] “Index of /genomes/all/GCA/000/001/635/GCA_000001635.9_GRCm39/GCA_000001635.9_GRCm39_assembly_structure/Primary_Assembly/assembled_chromosomes/FASTA.” https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/635/GCA_000001635.9_GRCm39/GCA_000001635.9_GRCm39_assembly_structure/Primary_Assembly/assembled_chromosomes/FASTA/ (accessed Feb. 15, 2023).

[20] P. Danecek *et al.*, “The variant call format and VCFtools,” *Bioinformatics*, vol. 27, no. 15, pp. 2156–2158, Aug. 2011, doi: 10.1093/BIOINFORMATICS/BTR330.

[21] D. J. Adams, A. G. Doran, J. Lilue, and T. M. Keane, “The Mouse Genomes Project: a repository of inbred laboratory mouse strain genomes,” *Mamm. Genome*, vol. 26, no. 9–10, pp. 403–412, Oct. 2015, doi: 10.1007/S00335-015-9579-6.

[22] W. McLaren *et al.*, “The Ensembl Variant Effect Predictor,” *Genome Biol.*, vol. 17, no. 1, pp. 1–14, Jun. 2016, doi: 10.1186/S13059-016-0974-4/TABLES/8.

[23] “API Documentation.” https://www.ensembl.org/info/docs/api/index.html (accessed Dec. 22, 2022).

[24] J. Navarro Gonzalez *et al.*, “The UCSC Genome Browser database: 2021 update,” *Nucleic Acids Res.*, vol. 49, no. D1, pp. D1046–D1057, Jan. 2021, doi: 10.1093/NAR/GKAA1070.

[25] R. Grantham, “Amino Acid Difference Formula to Help Explain Protein Evolution,” *Science (80-.).*, vol. 185, no. 4154, pp. 862–864, Sep. 1974, doi: 10.1126/SCIENCE.185.4154.862.

[26] Apple, “Overview · GitBook,” 2017. https://apple.github.io/turicreate/docs/userguide/ (accessed Jan. 20, 2023).

[27] “Documentation | Temporal Documentation.” https://docs.temporal.io/ (accessed Jan. 20, 2023).

[28] National Center for Biotechnology Information, “Mouse Genome Assembly GRCm39 - Genome Reference Consortium,” 2020. https://www.ncbi.nlm.nih.gov/grc/mouse/data (accessed Dec. 22, 2022).

[29] J. Wilson, J. M. Staley, and G. J. Wyckoff, “Extinction of chromosomes due to specialization is a universal occurrence,” *Sci. Reports 2020 101*, vol. 10, no. 1, pp. 1–13, Feb. 2020, doi: 10.1038/s41598-020-58997-2.

[30] “GRCm38 - mm10 - Genome - Assembly - NCBI.” https://www.ncbi.nlm.nih.gov/assembly/GCF_000001635.20/ (accessed Feb. 01, 2023).

[31] M. V. Volkenstein, “Probabilities of transversions and transitions (Russian),” *Mol. Biol.*, vol. 10, no. 4, pp. 737–741, 1976, Accessed: Dec. 22, 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/1023043/

[32] Z. E. Plyler, A. E. Hill, C. W. McAtee, X. Cui, L. A. Moseley, and E. J. Sorscher, “SNP Formation Bias in the Murine Genome Provides Evidence for Parallel Evolution,” *Genome Biol. Evol.*, vol. 7, no. 9, p. 2506, Sep. 2015, doi: 10.1093/GBE/EVV150.

[33] E. Schaper, O. Gascuel, and M. Anisimova, “Deep Conservation of Human Protein Tandem Repeats within the Eukaryotes,” *Mol. Biol. Evol.*, vol. 31, no. 5, pp. 1132–1148, May 2014, doi: 10.1093/MOLBEV/MSU062.

[34] F. Benachenhou *et al.*, “Evolutionary Conservation of Orthoretroviral Long Terminal Repeats (LTRs) and ab initio Detection of Single LTRs in Genomic Data,” *PLoS One*, vol. 4, no. 4, p. e5179, Apr. 2009, doi: 10.1371/JOURNAL.PONE.0005179.

[35] “Rab3il1 RAB3A interacting protein (rabin3)-like 1 [Mus musculus (house mouse)] - Gene - NCBI.” https://www.ncbi.nlm.nih.gov/gene/74760#gene-expression (accessed Mar. 08, 2023).

[36] “Population frequencies & genotypes.” https://www.ensembl.org/info/genome/variation/species/populations.html (accessed Mar. 07, 2023).

[37] “International Sheep Genomics Consortium.” https://www.sheephapmap.org/ (accessed Mar. 07, 2023).

[38] R. Lazarus, A. Kaspi, and M. Ziemann, “Creating reusable tools from scripts: the Galaxy Tool Factory,” *Bioinformatics*, vol. 28, no. 23, pp. 3139–3140, Dec. 2012, doi: 10.1093/BIOINFORMATICS/BTS573.

[39] E. Afgan *et al.*, “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2022 update,” *Nucleic Acids Res.*, vol. 50, no. W1, pp. W345–W351, Jul. 2022, doi: 10.1093/NAR/GKAC247.