PREDICTION OF RMSD OF ATOMIC POSITION IN CASP-5 PROTEIN STRUCTURE MODELS

COMPARISON OF GRADIENT BOOSTED REGRESSION TREE, FEEDFORWARD , AND GMDH DEEP NEURAL NETWORK METHODS

BY EVAN DOMINGOS

Abstract

We used 45730 observations from the 2002 Critical Assessment of Structure Prediction 5 Physiochemical Properties data set to train an optimized random forest regression model and deep neural network regression models using properties of amino acid residues to predict root-mean-square deviation of atomic positions (RMSD) for structural models. We investigate which physiochemical characteristics of amino acid residues are linked to higher error in the predicted tertiary structure and discuss the order of prediction dependence inherent to RMSD of structural models. Given the lack of order in the dataset, we found random forest and gradient boosted tree methods to most accurately predicted structural error which is heavily driven by polar surface area of residues.

Table of Contents

Abstract Cover

Introduction

Problem Statement

Basics of Protein Structure & Protein Folding

Calculation of RMSD

Pairwise Structural Alignment

CASP 5.9 Dataset

Preliminary Discussion

Methods

Random Forest Regression Tree

Feedforward Neural Networks

MIA GMDH Neural Network

Gradient Boosted Regression Tree

Discussion

Conclusion

Bibliography

Appendix

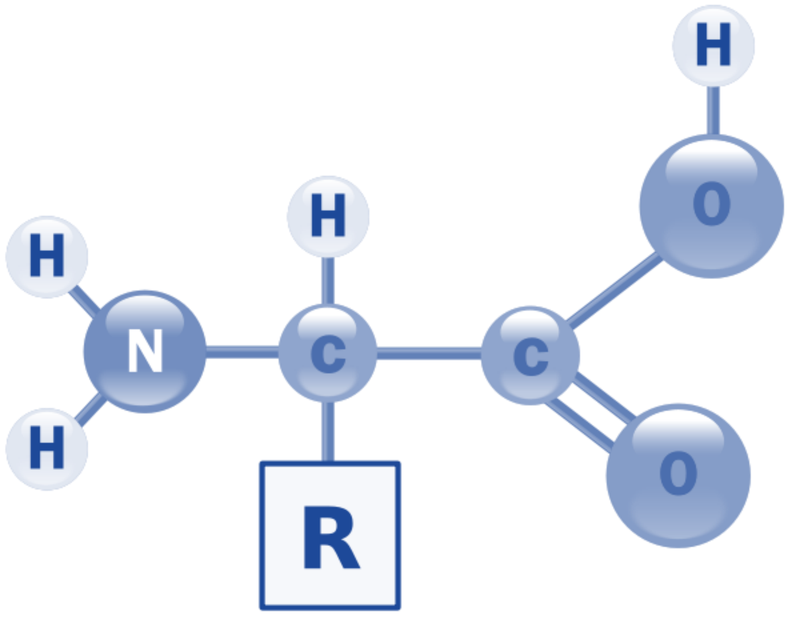
Introduction

Problem Statement

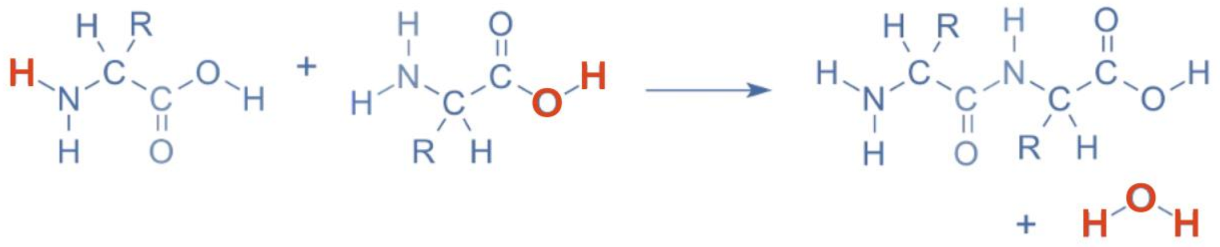
Proteins are vital to the functions of human cells, and they are responsible for performing a variety of tasks. There are tens of thousands of unique proteins within a typical human cell. Proteins fold into functional shapes and their different structures allow them to perform diverse roles. However, predicting the three-dimensional structure of a protein from its amino acid sequence remains a challenging task. Computational methods like machine learning algorithms have proven extremely well fitted for the task of protein folding predictions but it is crucial to identify and discard predicted models with structural errors, as relying on incorrect models can lead to erroneous conclusions about protein function and hinder the development of effective therapeutics. Some predictive models include predicted structural error factors to adjust predictions, similar to gradient boosting. We aim to construct a model to predict the structural errors in protein folding models from the 2002 CASP-5 competition.

Basics of Protein Structure & Protein Folding

Proteins are large biomolecules vital for performing functions within organisms like the replication of DNA, metabolizing enzymes, and structuring of cells. Proteins are formed from long chains of organic compounds called amino acids. Amino acids are formed around a single carbon atom called an alpha-carbon. Carbon has a valence of four so it may form four single bonds. In an amino acid, the alpha carbon is bonded to an amino group (NH2), a carboxyl group (COOH), a single hydrogen atom, and a variable element called the "R" group.



Two amino acids can form a peptide bond between the carboxyl group of one amino acid and the amino group of the other. A single water molecule (H2O) is formed from the expelled (H-) and (H0+), leaving a hydrophobic bond between the two amino acids called a peptide bond. The combined amino acids are called a polypeptide.



Any number of amino acids can bond to that structure, aligning the carboxyl group to the amino group of each successive amino acid resulting in a chain referred to as the primary structure of a protein. Two or more combined amino acids forming a peptide are also called a residue. There are 20 organically common amino acids that can be categorized by the chemical properties of their R-groups; polarity and electrical charge. The physical interaction of the R-group and the hydrogen bonds linking the amino acids influences the angle of the bond, contorting the flat primary structure into either a helical structure or a pleated sheet, named alpha helices and beta sheets respectively. This is referred to as the secondary structure. The tertiary structure is determined by the interaction of R-groups within the polypeptide chain.

These interactions include hydrogen bonding, ionic bonding, and hydrophobic interactions. The R-groups of amino acids that are hydrophilic (polar) tend to be located on the protein's surface, while those that are hydrophobic (non-polar) are generally buried in the structure's interior, away from the water. These physical interactions between the R groups of amino acids determine the protein's final three-dimensional shape, the tertiary structure which is ultimately responsible for determining the function of the resulting protein.

Primary Structure: Sequence of bonded amino acids (residues) forming a polypeptide
Secondary Structure: 3D structure from the interaction of R-groups and hydrogen bonds
Tertiary Structure: Contortions from the interaction of R-groups

Given the importance of protein function and structure in biochemistry for the development of drugs among other applications, the prediction of structure from amino acid sequences is of immense interest. Since the structures of proteins are determined by their amino sequences, protein structure prediction or protein folding has shown to be a powerful use-case for machine learning techniques and neural networks in particular. The use of machine learning algorithms in protein prediction has significantly reduced the time and cost required for the experimental determination of protein structures, making it a valuable tool in biochemistry research. Just 200,000 protein structures have been verified in the 130 years since x-ray crystallography was invented (Source: Protein Data Bank, 2023). In a single year, a neural network developed by Google’s DeepMind was able to produce and publish predictions for over 200 million known amino sequences with impressive results on verified structures.

This model named Alpha was developed for the Critical Assessment of Structure Prediction (CASP) which is a biennial worldwide experiment to evaluate the efficacy of computational protein structure prediction methods developed by academia and industry teams. CASP allows researchers from around the world to test their prediction methods on a set of protein sequences whose structures have not been determined experimentally or whose verified structures have not been released.

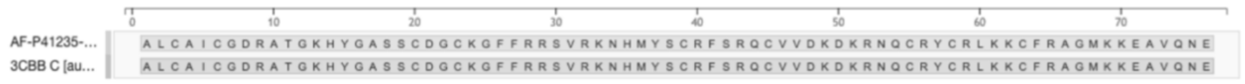
The scoring data from the 2002 CASP 5 competition was published to the UCI ML Database which was used for this project. The data contains residue prediction scores for models submitted for the competition, measuring the deviation in position between the predicted structure and the aligned verified protein structure.

Calculation of RMSD

Protein prediction models are evaluated on the residuals of the predicted structure and the verified structure. Results in the CASP 5.9 dataset are evaluated using root mean squared deviation of atomic position (RMSD).

Both modeled and verified structures have the same spatial geometry. The structure is indexed by the amino sequence. For the purpose of prediction scoring, the sequences are identical.

Example:



```
                  0         10        20        30        40        50        60        70
AF-P41235-...   A L C A I C G D R A T G K H Y G A S S C D G C K G F F R R S V R K N H M Y S C R F S R Q C V V D K D K R N Q C R Y C R L K K C F R A G M K K E A V Q N E
3CBB C [au...   A L C A I C G D R A T G K H Y G A S S C D G C K G F F R R S V R K N H M Y S C R F S R Q C V V D K D K R N Q C R Y C R L K K C F R A G M K K E A V Q N E
```

Source: *Protein Data Bank*

The alpha carbon for a given residue is treated as the location for that amino acid and each residue in the structural model has XYZ coordinates. Generally speaking, the XYZ coordinates treat the center of the structure or sequence as the origin. The position of following amino acids are dependent on the position of the preceding residue in the sequence.
Root mean squared deviation is calculated at each residue for the difference between the predicted and verified locations of the alpha carbon.

The general equation for RMSD of atomic position in an XYZ coordinate system is:

$$RMSD\ (v,w) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}((v_{ix}-w_{ix})^2+(v_{iy}-w_{iy})^2+(v_{iz}-w_{iz})^2)}$$

where n is the number of points in the measurement (residues in the structure where v is the verified location and w is the predicted location. In the context of this paper, observations are scores for single residues thus we can consider that n = 1 for all measurements and the formula for RMSD is equivalent to the distance formula in a 3-dimensional cartesian coordinate system.

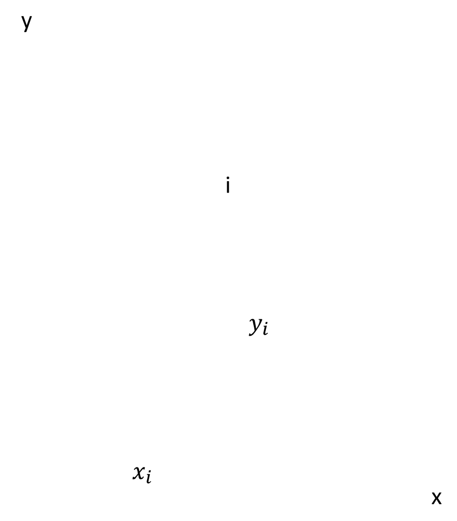
$$RMSD\ (v,w) = \sqrt{(v_x-w_x)^2+(v_y-w_y)^2+(v_z-w_z)^2}$$

Again, v is the verified alpha carbon location and w is the predicted location. RMSD is reported in Angstroms (A) where 1A = 10E-10 meters.

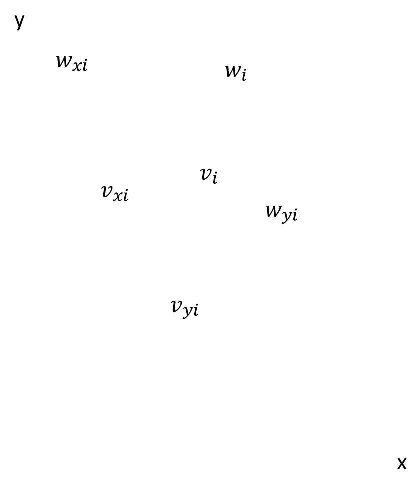
Pairwise Structural Alignment

Protein structures are compared via pairwise structural alignment. In the context of model validation, the verified structure of the protein is suspended in a 3-dimensional coordinate system. The predicted structure is superimposed in the space. In rigid body alignment which was used in CASP-5, the relative orientations and positions of the alpha carbons in each structure remain fixed through the alignment process (Bank, n.d.). The structural model of each protein is considered as a matrix for which the Kabsch algorithm (or an estimation of the algorithm) is employed to calculate the optimal rotation matrix to minimize the distance or RMSD between the two. The algorithm first translates each structure so that the geometric center or the centroid is positioned at the origin of the 3-dimensional coordinate system. The algorithm then calculates the rotation matrix so as to minimize the objective function of RMSD (Kabsch, 1976).

Structural geometry in a 2-dimensional coordinate system

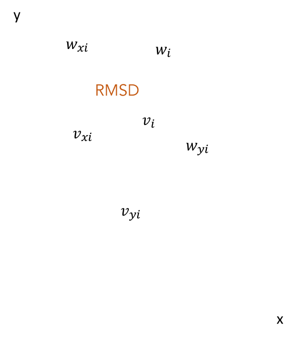


Pairwise Structural Alignment



v is the verified structure centered at the origin and w is the superimposed predicted structure

Calculation of RMSD for the ith residue in the aligned structures



CASP 5.9 Dataset

The CASP 5.9 Dataset contains 45730 observations of scored results from the 2002 CASP 5 competition. 67 newly validated proteins where used to score structure predictions of the submitted models. This dataset is ***not*** the structural predictions themselves. Each observation represents a prediction for a single amino acid and contains physiochemical properties from the prediction as well as RMSD. No information for the specific protein, amino acid, or predictive model is provided in the data set.

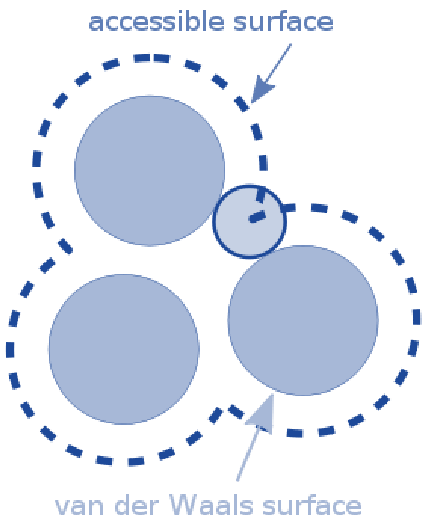
Further, it is not clear the number of unique residues contained within the score (multiple models making predictions for the same protein). The CASP-5 validation set (used in the competition, not the UCI data set) contained a total of 14882 residues for 67 different proteins. The UCI dataset contains scores for 45730 residues so at a minimum, the dataset contains predictions from multiple models for the same amino residues.

The dataset has 9 features and the label RMSD. The 9 features are all continuous numerical variables defined as the following:

F1 - Total surface area (Angstroms
F2 - Non-polar exposed area
F3 - Fractional area of exposed non-polar residue.
F4 - Fractional area of exposed non-polar part of residue
F5 - Molecular mass weighted exposed area.
F6 - Average deviation from the standard exposed area of residue.
F7 - Euclidean distance.
F8 - Secondary structure penalty.
F9 - Spatial Distribution constraints (N,K Value).

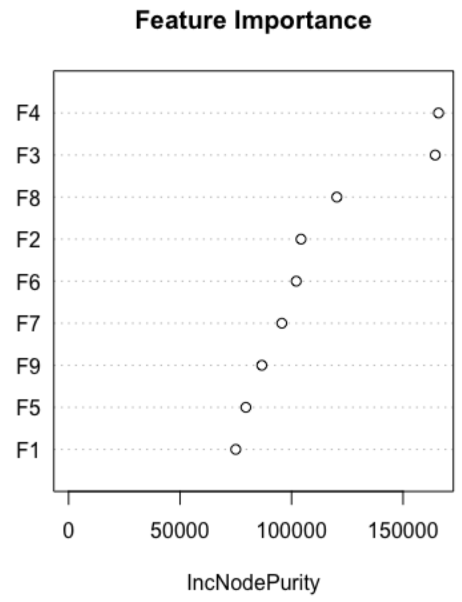
Exposed or accessible surface area (ASA) is the surface area of the protein that is accessible to a solvent, water in this application. One method of calculating ASA is the Shrake-Rupley algorithm which draws a mesh of points around each atom of the residue with a radius equal to the Van der Waals radius (conceptually the surface of the atom) plus the radius of the solvent (water is 1.4Å) (Ribeiro et al., 2019). In essence, the algorithm rolls a ball the size of a water molecule along the surface of the residue.

Visual of Accessible Surface Area Calculation

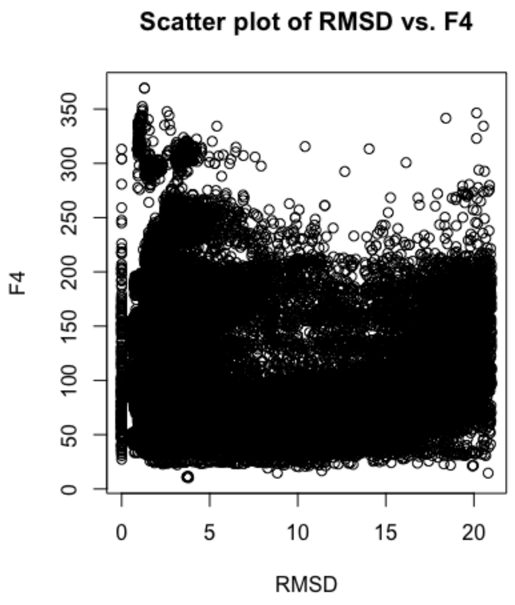


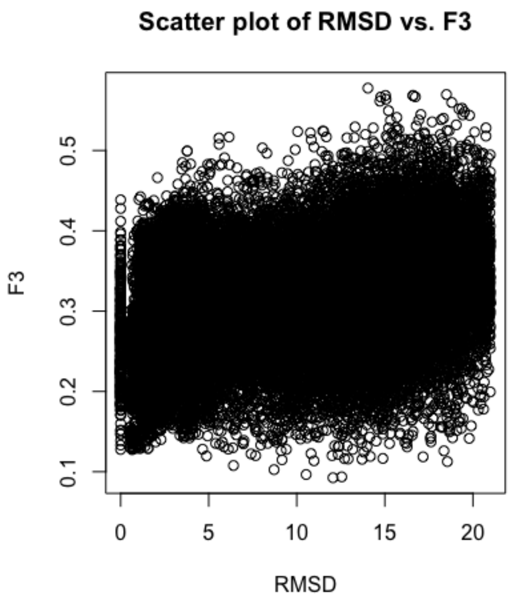
Preliminary Discussion

There is no visible structure between RMSD and any of the predictors. To assess the relative importance of each feature we utilized a random forest regression tree model on all features. The architecture of the model is discussed in the Methods section. Below is the feature importance of the trained model.



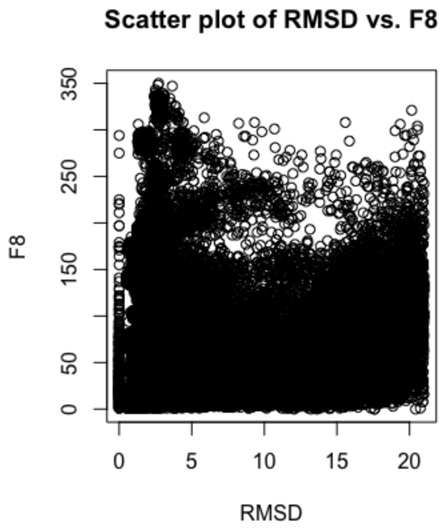
F4 and F3 are Fractional area of exposed non-polar part of residue and Fractional area of exposed non-polar residue respectively. The importance of these features was consistent with all interpretable models. In the context of protein folding physical principles, the importance is explained in that non-polar regions are hydrophobic. Water is a chemical product of protein folding and the repulsive interaction between water and the hydrophobic surface causes the protein to contort so that the non-polar residues are inward facing. In the context of modeling structure, more complicated physicochemical interactions would be positively correlated with error.





Note the clustering in the plot of RMSD vs. F4 and the loose positive correlation between F1 and RMSD

F8, the Secondary structure penalty is not clearly defined in the UCI dataset but appears to be related to the dehydration process (production of water) and the resulting formation of a hydrogen bond which is also non-polar. F2 is the non-polar exposed area, similar to the above features (Yasuda et al., 2010).



Note the clustering in the plot of RMSD vs. F8, similar to that of F4

Features pertaining to total surface area (F1), mass-weighted exposed area (F5), and Euclidean distance (F7) are highly colinear and seem to correlate with the size of the residue. There is no apparent conceptual argument for any relationship between the size of the residue and the predicted aligned error.

Models & Results

All models in this analysis are written in R. All R code as well as some Python rewrites can be found in the appendix or at https://github.com/evandomingos/CASP-Protein-Prediction

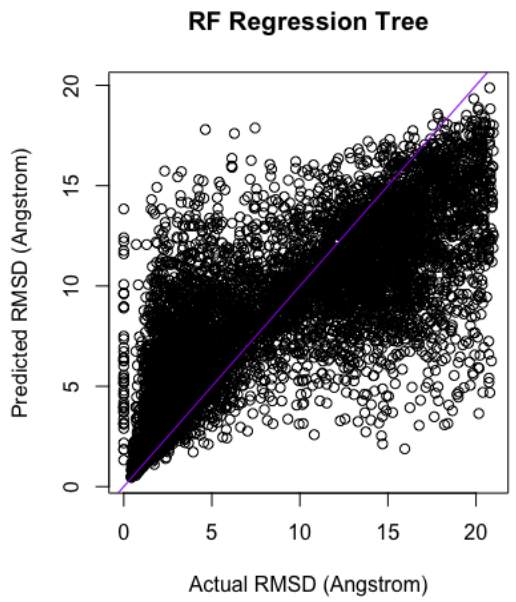
Random Forest Regression Tree

The first method analyzed was a random forest regression model. A different setup for training and validation sets was use for this model where 60% of the cleaned protein dataset was included in the training and the remaining 40% was broken up into two validation sets. The objective of this model was to overfit in an attempt to minimize error on the training set. The features and label are not scaled.

Random Forest model

```
rf_model <- randomForest(train_x, train_y, ntree = 500)
```

The model was trained with ntree = 500. No tangible gain in accuracy on the test set was seen for n>500.



The trained model reported a validation set MAE of 2.540 Angstroms. Despite minsplit = 1 which is prone to overfitting, the bagged model of trees was one of the best performers, especially given time to train. The model generally overpredicts lower RMSD and underpredicts higher values.

Feedforward Neural Network

Several carryforward neural networks were developed with PyTorch, all of which are contained in the appendix. The input data for all neural networks was scaled and the intercept removed as follows:

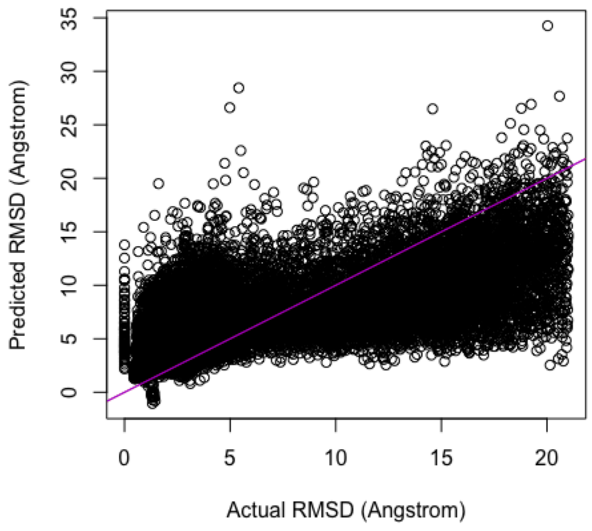
```
x <- scale(model.matrix(protein$RMSD ~ . - 1, data = protein))
y <- protein$RMSD
```

torch_manual_seed was set = 7 for all trained models.

modNN
The first model tested, modNN1, had a simple architecture of a single hidden layer with 50 neurons and used a rectified linear unit (ReLU) activation function. Changing the number of neurons in the hidden layer did not have a tangible impact on the convergence speed or trained model. The model also contained a single dropout layer for regularization where the dropout

rate was set to 0.4. Changing the dropout rate to <0.3 or >0.5 resulted in a model that failed to converge in the allowed training period. This model was trained for 10 epochs using the mean squared error (MSE) loss function.

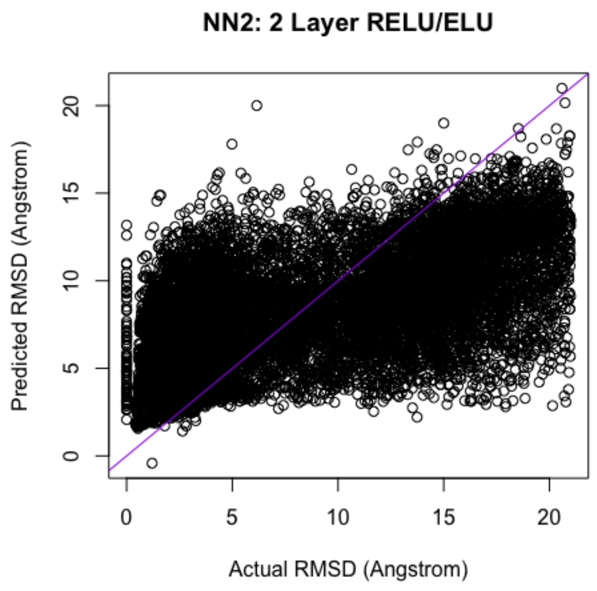
```
modnn <- nn_module(
  initialize = function(input_size) {
    self$hidden <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout <- nn_dropout(0.4)
    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    ## x %>%
    ##   self$hidden() %>%
    ##   self$activation() %>%
    ##   self$dropout() %>%
    ##   self$output() %>%
    self$output( self$dropout( self$activation1( self$hidden(x) ) ) )
  }
)

#set hyperparameters for nn_module
# MSE loss function
modnn <- set_hparams( setup(modnn ,
                            loss = nn_mse_loss(),
                            optimizer = optim_rmsprop,
                            metrics = list(luz_metric_mae())),
                      input_size = ncol(x))


#training
#takes roughly 6 hours to train at 100 epochs, 210 sec each
fitted <-   fit(modnn,
                data = list(x[-testid, ],
                            matrix(y[-testid], ncol = 1)),
                valid_data = list(x[testid, ],
                                  matrix(y[testid], ncol = 1)),
                epochs = 10)
```



Training MAE: 3.89 Angstroms
Validation MAE: 4.56 Angstroms

In training, this model was a poor learner and failed to advance MAE meaningfully with further training time.

modNN2
The second model "modnn2," is an iteration of the modnn architecture, with the addition of a second hidden layer with 50 neurons that uses the ELU activation function. The size of the hidden layer was kept the same and the model was trained for 100 epochs. This model was very sensitive to the learning rate which was found to be optimal for convergence at 0.4. IF this value was changed +/- 0.1, the model failed to converge entirely.



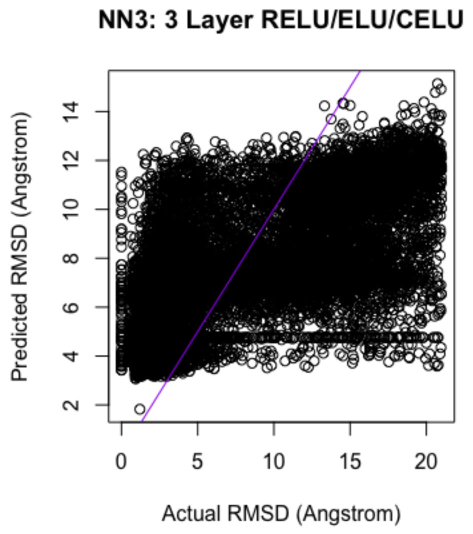
Training MAE: 3.89 Angstroms
Validation MAE: 4.56 Angstroms

modNN3
The "modnn3" model is identical to the second but with the addition of a hidden layer that uses the continuously differentiable exponential linear unit (CELU) activation function of the form:

$CELU(x)=max(0,x)+min(0,\alpha*(exp(x/\alpha)-1))$

This model was also trained for 100 epochs and reported the following accuracy metrics:

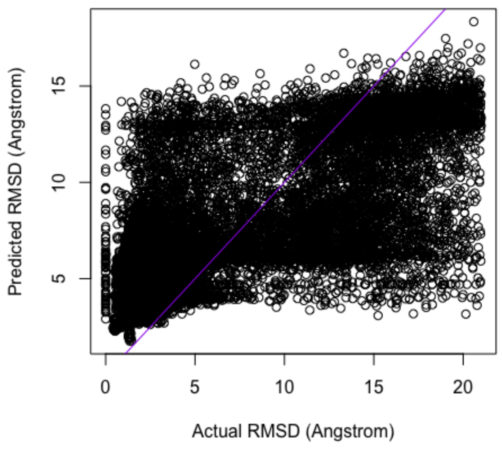
Training MAE: 2.66213 Angstrom
Validation MAE: 3.2498 Angstrom



This addition of the CELU layer model markedly improved predictive power of the model and the residuals showed some structure. The model had an issue with predictions RMSD at 5 Angstroms where there's a line segment of 5 Angstrom predicted values evenly distrivuted by their actual RMSD.

modNN5
The fifth model is a neural network with twelve hidden layers, alternating in ReLU and ELU activation functions. The output layer has one neuron and uses a linear activation function. The presented model uses the MSE as the loss function, the RMSprop optimizer for optimization, and the MAE as the evaluation metric. A variation utilizing dataloader and the SGD optimization algorithm was implemented but was not used to produce the presented results.
The dropout rates for the hidden layers are set to 0.5 for all layers. Multiple dropout rates were evaluated. A dropout rate of 0.4-0.5 resulted in optimal convergence. It is possible

Training MAE: 2.68
Validation MAE: 2.98



MIA GMDH Neural Network

modNN7 is a Multilayered Iterative (MIA) Group Method of Data Handling (GMDH) Neural Network, implemented using the gmdh.mia() function from the GMDHreg library in R.

The GMDH (Group Method of Data Handling) algorithm is a neural network architecture that involves iteratively generating candidate models and selecting the best one based on some criterion to create a model of models. In this case, the MIA-GMDH algorithm is used, which generates candidate models and selects the best model based on the predicted residual error of sum squares (PRESS). Nodes are pruned from each layer and the top k models are selected.

Prune (an arugment in the gmdh.mia() function) is the selected number of neurons from layer i to layer i+1. The resulting layer i+1 has prune(prune-1)/2 neurons. The selected models are then used to generate a set of candidate models for the second layer, where each candidate model consists of the combination of the inputs from the selected models in the first layer. This process is repeated for each subsequent layer until the final layer is reached.

The output of one layer is used as input to the subsequent layer, resulting in a hierarchical stack of models.

.

For modnn7, the input data for the model is identical to the previous carryforward neural networks but must be treated as a matrix object of predictor variables (x) and a vector of the response variable (y).

The prune argument specifies the maximum number of predictors to include in each layer of the model, while the criteria argument specifies the criterion to use for selecting the best model. In this model, prune is specified as ncol(x) or 9, as per recommended in the GMDHreg documentation.

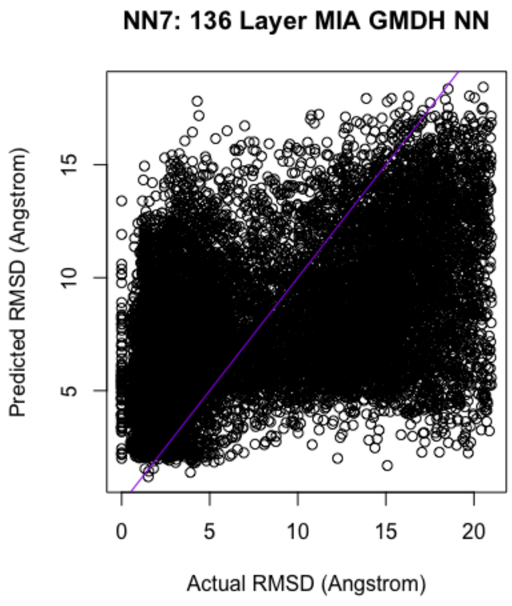
In this case, the PRESS (Predicted Residual Error Sum of Squares) criterion is used, which considers all information in the data sample and is computed without recalculation at each test point. “test” is an estimation RMSE and is more computationally efficient than “PRESS”. PRESS is used as the stopping criterion in the training process and can be thought about as ridge regression in that it adds a penalty for additional complexity to the model. The algorithm continuously adds hidden layers until this stopping criterion is met.

modNN7

```
# Fit GMDH NN model
modnn7 <- gmdh.mia(X = as.matrix(protein[-testid, 2:10]),
                      y = as.matrix(protein[-testid, 1]),
                      prune = ncol(protein[-testid, 2:10]),
                      criteria = "PRESS")
```

The trained modNN7 model using set.seed(13) produced a 136 hidden layer model. Notably, this model was a significantly faster learner than the strict carryforward neural networks.

When making predictions on the validation set, the model outputted 27 missing (NA) values. These values were removed using row indexing. Finally, the mean absolute error (MAE) is calculated on the test set (protein.test[,1]) using the cleaned predicted values.



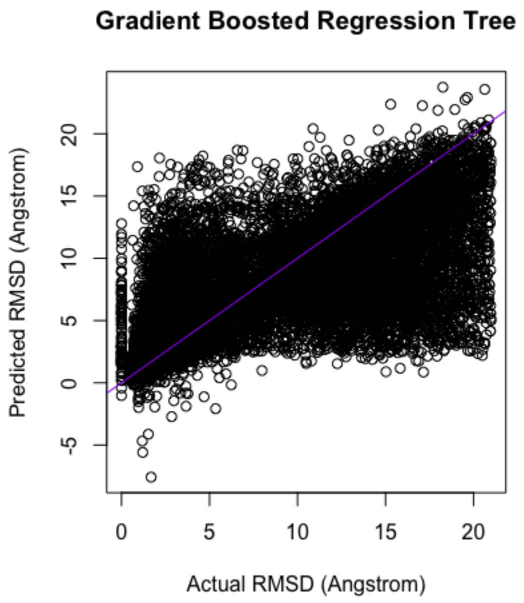
Gradient Boosted Regression Tree

The final method evaluated was a gradient-boosted regression tree using the XGBoost R library. Gradient boosted trees are essentially an ensemble model made of successive decision trees starting by building a simple regression tree. The algorithm then calculates the residuals on each training data point and trains a second tree to predict those residuals from the first tree. This process is iterated for n-rounds where each successive tree predicts the residuals from the previous. After n-rounds, the final model is the ensemble of all trees. An outputted prediction is the sum of the prediction from all n-trees in the model.

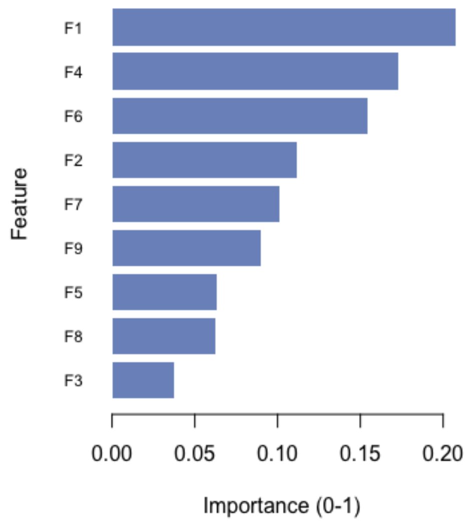
$$\hat{Y} = \hat{Y}_1 + \hat{\varepsilon}_2 + \hat{\varepsilon}_n + \cdots + \hat{\varepsilon}_n$$

Where $\hat{Y}_1$ is the predicted value from the initial tree and $\hat{\varepsilon}_n$ is the predicted residual from the nth tree

The objective argument of the gradient-boosted regression algorithm determines how the residuals are calculated and how the algorithm tries to minimize them. For this model, the objective function used is the mean absolute error (reg:absoluteerror) which is minimized via gradient descent.



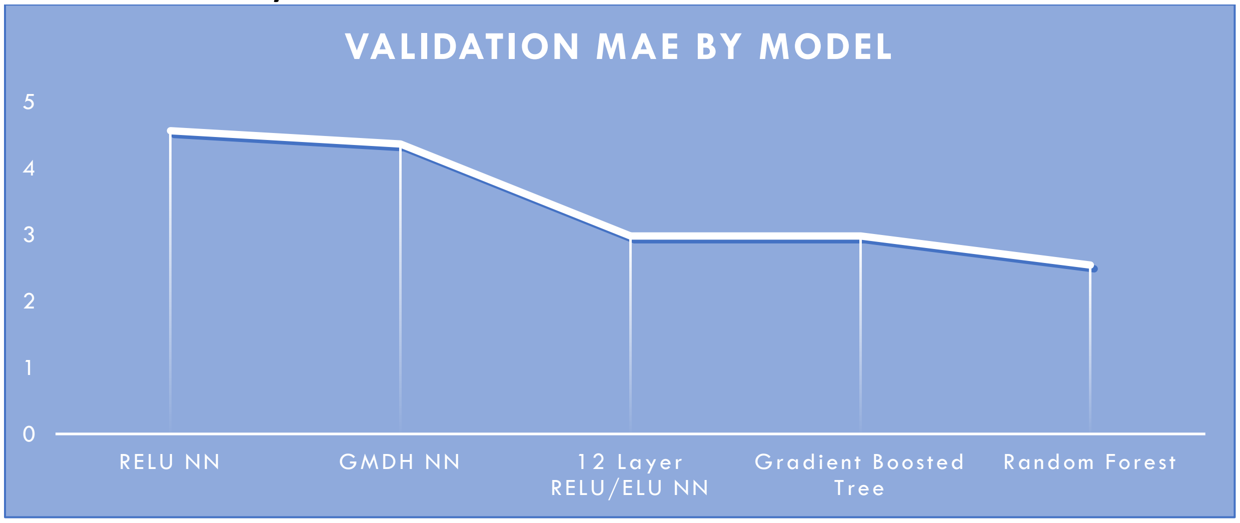
The gradient-boosted tree model trained nrounds = 10,000 and descended to an MAE of 2.628 Angstrom on the training set. The trained model produced an MAE of 2.979 Angstrom on the test set which was markedly better than the neural network models. This model could be improved returning predicted negative values as zero.



More interesting was the feature importance of the Gradient Boosted model. F1, total exposed surface area, which was the least important feature of the random forest model was the heaviest weighted feature through the ensemble of models followed by F4 which is a non-polar surface area feature. Given the importance of non-polar regions to the initial regression tree, the wight of F1 can be interpreted as predictive of the residual.

Discussion

Validation Results by Model



The random forest and gradient-boosted tree proved the best methods for the prediction of RMSD considering both overall accuracy as assessed by MAE, the time to train the model, and overall interpretability.

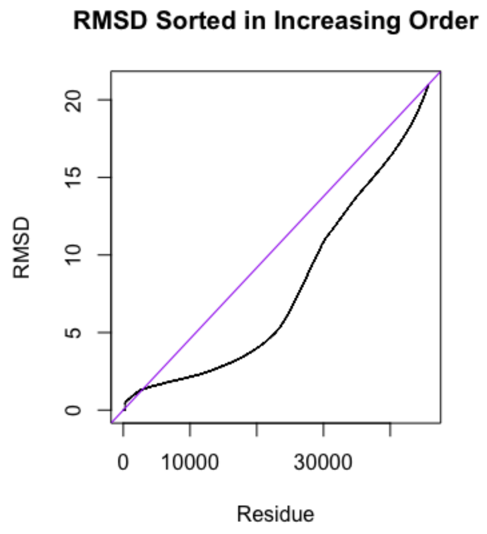
The carryforward neural network architectures proved were considerably slower to train and failed to minimize MAE to the same extent. The addition of incremental hidden layers improved the accuracy of the model with a diminishing return on complexity.

The GMDH method was not trained to the extent of the carryforward nueral networks. In the time allowed, this architecture proved to be a considerably better learner and was faster to pick up structure in the dataset. The model present could be improved upon by increasing the prune setting which exponentially increases the size of the neural network. For the purpose of this study, a prune setting much larger than 9 was too computationally expensive with the hardware available. A similar GMDH architecture with the GIA algorithm is outlined in the appendix.

Overall, all residuals for any of the presented models are too high to be of significant use. MAE of 3 Angstrom is the van der Waals diameter of an entire water molecule which means we cannot predict with meaningful certainty error in a predicted structure. Even allowing for overfitting, the floor for MAE appeared to be roughly 2.5 Angstrom.

Returning to the fundamentals of RMSD computation and structural predictions, it appears this error inherent to the dataset is caused by the dependence of order for RMSD. The predicted location of a residue is a product of the predicted location of the preceding residue and the relative orientation of it's chemical bond. If the preceding residue has a high error, the following residue can have an identical RMSD even if the prediction of its bond angle has no error because the predicted location is relative to the preceding residue.

Below is a plot of RMSD from the original protein dataset where RMSD is sorted by increasing order. Several entries are zero or near zero such that the predicted residues are aligned with the validated location. The increasing slope of RSMD appears to agree with the notion of location dependence.



Conclusion

Our results suggest that random forest and gradient-boosted trees are the best methods for predicting RMSD of those explored, as they achieved the highest accuracy with reasonable training time and interpretability. On the other hand, carryforward neural networks were considerably slower to train and failed to minimize MAE to the same extent. The GMDH method, while not trained extensively, showed promise in being a fast learner and was faster to pick up structure in the dataset. However, all the presented models had residuals that were too high to be of significant use, clearly indicating that predicting the structure of proteins with meaningful certainty remains a challenge and the dataset sued lacked features that captured the dependence of order for RMSD which poses a challenge to further work with this dataset. Overall, this project provided valuable insights into the performance of various machine learning methods as well as the fundamentals of protein structures and biochemistry.

Bibliography

Admin. (2014, January 9). Protein Folding: The Good, the Bad, and the Ugly - Science in the News. Science in the News. https://sitn.hms.harvard.edu/flash/2010/issue65/

Bank, R. P. D. (n.d.). Pairwise Structure Alignment. https://www.rcsb.org/docs/tools/pairwise-structure-alignment

- DECOYSETS2019. (n.d.). https://predictioncenter.org/decoysets2019/description.cgi?casp=CASP5

Gloyd, J. (2023). Protein Folding. ChemTalk. https://chemistrytalk.org/protein-folding/

Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. Acta Cryst. A32, 922–923.

Ribeiro, J., Ríos-Vera, C., Melo, F. S., & Schüller, A. (2019). Calculation of accurate interatomic contact surface areas for the quantitative analysis of non-bonded molecular interactions.

Bioinformatics, 35(18), 3499–3501. https://doi.org/10.1093/bioinformatics/btz062 Stepashko, V., Bulgakova, O., & International Research and Training Centre for Information Technologies and Systems of the NAS and MES of Ukraine. (2013). Generalized Iterative Algorithm GIA GMDH. International Conference in Inductive Modelling ICIM'.

Tilve, M. V. (2021a, July 5). GMDHreg: an R Package for GMDH Regression. https://cran.r-project.org/web/packages/GMDHreg/vignettes/GMDHreg.html

Tilve, M. V. (2021b). GMDHreg: Regression using GMDH Algorithms. rdrr.io. https://rdrr.io/cran/GMDHreg/

UCI Machine Learning Repository: Protein Data Data Set. (n.d.). https://archive.ics.uci.edu/ml/datasets/Protein+Data

Yasuda, S., Yoshidome, T., Oshima, H., Kodama, R., Harano, Y., & Kinoshita, M. (2010). Effects of side-chain packing on the formation of secondary structures in protein folding. Journal of Chemical Physics, 132(6), 065105. https://doi.org/10.1063/1.3319509

Appendix

Plots

```
dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/00265/CASP.csv", header = TRUE,
stringsAsFactors = FALSE)

library(ggplot2)
#remove missing data (none in CASP dataset), create protein set
protein <- na.omit(dataset)
names(protein)

vars <- names(protein)
combinations <- combn(vars, 2)
#scatter plot each variables
par(mfrow = c(1, 1))
par(ask = TRUE)
for (i in 1:ncol(combinations)) {
  plot(protein[, combinations[1, i]], protein[, combinations[2,
i]],
       xlab = combinations[1, i], ylab = combinations[2, i],
       main = paste0("Scatter plot of ", combinations[1, i], "
vs. ", combinations[2, i]))
}

# sort by increasing RMSD values
protein_sorted <- protein[order(protein$RMSD),]
# plot RMSD in increasing order
```

```
plot(protein_sorted$RMSD, type = "l", xlab = "Residue", ylab =
"RMSD", main = "RMSD Sorted in Increasing Order")
abline(a = 0, b = 20.99/45730, col = "purple")
par(ask = FALSE)
```

Models

```
#Various libraries
library(tidyverse)
library(caret)
library(tensorflow)
library(ranger) # Random forest with Ranger (more efficient than
rf library for higher dimensional data)
library(randomForest) # Random Forest
library(xgboost) # For Gradient Boosted Tree
library(nnet)  # Neural net
library(keras)
library(gtable)
library(zeallot)
library(rpart)
library(rpart.plot)
library(torch)
library(luz) # High-level interface for torch
library(torchvision) # For datasets and image transformation
library(GMDHreg) # Library for Group Method of Data Handling
(GMDH) Nueral Network

# Load CASP 5.9 Set from
dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/00265/CASP.csv", header = TRUE,
stringsAsFactors = FALSE)

# Remove missing data (none in CASP dataset), create protein set
protein <- na.omit(dataset)

#n number of protein residues: 45730 obs
n <- nrow(protein)
# For reproduction
set.seed(7)

# Number of test samples = 1/3 observations
ntest <- trunc(n / 3)
#randomly select test IDs from
testid <- sample(1:n, ntest)

# Set PyTorch seed
torch_manual_seed(7)
```

```
# In this segment,  only Gradient Boosted tree uses
# Scale variables, separate x labels ,y target var. Remove
intercepts
x <- scale(model.matrix(protein$RMSD ~ . - 1, data = protein))
length(x)
y <- protein$RMSD







# First Method
# Random Forest regression tree with the obejctive of plotting
feature important and purposely overfitting a model
# Very easy to produce good looking tree, extremely overfitted
and validation MAE explodes

library(randomForest)


# Split the data into training and testing sets

# *** Un-comment these blocks to run the Random Forest Model.
Commented out so the train_id object does not affect Nueral
Networks ***

# For reproducibility
set.seed(13)
#train_id <- sample(1:n, floor(n * 0.6), replace = FALSE)
#valid_id <- sample(setdiff(1:n, train_id), floor(n * 0.2),
replace = FALSE)
#test_id <- setdiff(1:n, union(train_id, valid_id))

# Create extra holdout set, "valid"
#train_x <- x[train_id, ]
#train_y <- y[train_id]
#valid_x <- x[valid_id, ]
#valid_y <- y[valid_id]
#test_x <- x[test_id, ]
#test_y <- y[test_id]


# Train the random forest regression model
rf_model <- randomForest(train_x, train_y, ntree = 500)
```

```
# Plot feature importance
varImpPlot(rf_model, main = "Feature Importance")

# Predict RMSD for test set using the trained model
pred_valid <- predict(rf_model, valid_x)
pred_test <- predict(rf_model, test_x)

# Calculate mean squared error on the test set and validation
(extra holdout data)
mae_valid <- mean(abs(pred_valid - valid_y))
mae_test <- mean(abs(pred_test-test_y))



# Scatterplot of predicted vs actual RMSD on validation set
plot(valid_y, pred_valid, xlab = "Actual RMSD (Angstrom)", ylab
= "Predicted RMSD (Angstrom)", main = "RF Regression Tree")
abline(a = 0, b = 1, col = "purple")

# Scatterplot of predicted vs actual RMSD on test set
plot(test_y, pred_test, xlab = "Actual RMSD (Angstrom)", ylab =
"Predicted RMSD (Angstrom)", main = "RF Regression Tree")
abline(a = 0, b = 1, col = "purple")

# MAE 2.5555 on test set
# After purposely overfitting as much as possible (try
nrounds >>500) this MAE of 2.55 appears to be the absolute
minimum error achievable
mae_test

# MAE 2.541 on validation, second holdout set
mae_valid


# Conventional Regression Tree for the purpose of displaying
featurew importance
protein_tree1 <- rpart(RMSD ~ ., data = protein[train_id,],
control = rpart.control(minsplit = 1))

# Predict on Training set
train_pred_protein_tree1 <- predict(protein_tree1,
protein[train_id,])
# Calculate MAE on training set
train_tree1_mae <- mean(abs(train_pred_protein_tree1-
protein$RMSD[train_id]))
# Train MAE 5.49
```

```
train_tree1_mae

# Predict on Test Set, Calculate MAE
test_pred_protein_tree1 <- predict(protein_tree1, newdata =
protein[test_id, ])

# Calculate MAE for test set
mae_protein_tree1 <- mean(abs(pred_protein_tree1 - y[testid]))
# Test MAE 5.49
mae_protein_tree1

# Plot Fitted Tree
rpart.plot(protein_tree1)

set.seed(123)
# Gradient Boosted Tree
# Gradient Boosted Regression Tree with MAE Objective Function
gdboosteds_protein <- xgboost(data = x[-testid,], label = y[-
testid], nrounds = 20000, objective = "reg:absoluteerror")
gdboost_test_pred <- predict(gdboosteds_protein, x[testid, ])
# For set.seed(13), nrounds = 10,000, minsplit = DEFAULT,
trained MAE = 2.628

# Calculate MSE
mse <- mean((gdboost_test_pred-y[testid])^2)
# 19.3254 MSE for set.seed(13), nround = 10,000
mse


# Calculate MAE on test set
mae <- mean(abs(gdboost_test_pred-y[testid]))
# 2.979239 MAE for set.seed(7), nround = 10000
mae

# Scatterplot of predicted vs actual RMSD
plot(y[testid], gdboost_test_pred, xlab = "Actual RMSD
(Angstrom)", ylab = "Predicted RMSD (Angstrom)", main =
"Gradient Boosted Regression Tree")
abline(a = 0, b = 1, col = "purple")


importance_matrix <- xgb.importance(model = gdboosteds_protein)
importance_matrix
xgb.plot.importance(importance_matrix, col =
rgb(124/256,148/256,198/256), xlab = "Importance (0-1)",
                    ylab = "Feature")
```

```
# Load CASP 5.9 Set from
dataset <- read.csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/00265/CASP.csv", header = TRUE,
stringsAsFactors = FALSE)

# Remove missing data (none in CASP dataset), create protein set
protein <- na.omit(dataset)

# n number of protein residues: 45730 obs
n <- nrow(protein)
# For reproduction
set.seed(7)

# Number of test samples = 1/3 observations
ntest <- trunc(n / 3)
#randomly select test IDs from
testid <- sample(1:n, ntest)

# Set PyTorch seed
torch_manual_seed(7)

# Scale variables, separate x labels ,y target var. Remove
intercepts
x <- scale(model.matrix(protein$RMSD ~ . - 1, data = protein))
y <- protein$RMSD


# PYTORCH NEURAL NETWORKS

#create NN with four layers, input, 50 node hidden layer, Relu
activation, dropout layer 0.4
modnn <- nn_module(
  initialize = function(input_size) {
    self$hidden <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout <- nn_dropout(0.4)
    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    ## x %>%
    ##   self$hidden() %>%
    ##   self$activation() %>%
    ##   self$dropout() %>%
    ##   self$output() %>%

self$output( self$dropout( self$activation1( self$hidden(x) ) )
)
  }
)

#set hyperparameters for nn_module
# MSE loss function
modnn <- set_hparams( setup(modnn ,
                            loss = nn_mse_loss(),
                            optimizer = optim_rmsprop,
                            metrics = list(luz_metric_mae())),
                      input_size = ncol(x))


#training
#takes roughly 6 hours to train at 100 epochs, 210 sec each
fitted <-   fit(modnn,
                data = list(x[-testid, ],
                            matrix(y[-testid], ncol = 1)),
                valid_data = list(x[testid, ],
                                  matrix(y[testid], ncol = 1)),
                epochs = 10)


plot(fitted)
summary(fitted)

fitted$ctx

# Predict RMSD values for test set
test_pred <- predict(fitted, x[testid, ])

# Calculate MSE on test set
mse <- mean((test_pred-y[testid])^2)
mse
# MSE of 50.7419 for 100 epoch model


# Scatterplot of predicted vs actual RMSD
```

```
plot(y[testid], test_pred, xlab = "Actual RMSD (Angstrom)", ylab
= "Predicted RMSD (Angstrom)", main = "RELU HL Nueral Network")
abline(a = 0, b = 1, col = "purple")



# 2nd Model evalauted, Addition of ELU hidden layer
modnn2 <- nn_module(
  initialize = function(input_size) {
    self$hidden1 <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout1 <- nn_dropout(0.4)
    self$hidden2 <- nn_linear(50, 50)
    self$activation2 <- nn_elu()
    self$dropout2 <- nn_dropout(0.4)
   # self$hidden3 <- nn_linear()

    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation1() %>%
      self$dropout1() %>%
      self$hidden2() %>%
      self$activation2() %>%
      self$dropout2() %>%
      self$output()
  }
)

modnn2 <- set_hparams( setup(modnn2 ,
                            loss = nn_mse_loss(),
                            optimizer = optim_rmsprop,
                            metrics = list(luz_metric_mae())),
                      input_size = ncol(x))


# Training for ModNN2
#210sec per epoch
fitted2 <-   fit(modnn2,
                data = list(x[-testid, ],
                            matrix(y[-testid], ncol = 1)),
                valid_data = list(x[testid, ],
                                  matrix(y[testid], ncol = 1)),
                epochs = 1)
```

```
plot(fitted2)
summary(fitted2)

fitted2$ctx

#predict RMSD values for test set
test_pred2 <- predict(fitted2, x[testid, ])
mae_nn2 <- mean(abs(test_pred2 - y))
mae_nn2
plot(y[testid], test_pred2, xlab = "Actual RMSD (Angstrom)",
ylab = "Predicted RMSD (Angstrom)", main = "NN2: 2 Layer
RELU/ELU")
abline(a = 0, b = 1, col = "purple")




#ModNN3

#add CELU hidden layer
modnn3 <- nn_module(
  initialize = function(input_size) {
    self$hidden1 <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout1 <- nn_dropout(0.4)
    self$hidden2 <- nn_linear(50, 50)
    self$activation2 <- nn_elu()
    self$dropout2 <- nn_dropout(0.4)
    self$hidden3 <- nn_linear(50, 50)
    self$activation3 <- nn_celu()
    self$dropout3 <- nn_dropout(0.4)
    # self$hidden3 <- nn_linear()

    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation1() %>%
      self$dropout1() %>%
      self$hidden2() %>%
      self$activation2() %>%
      self$dropout2() %>%
      self$hidden3() %>%
      self$activation3() %>%
      self$dropout3() %>%
      self$output()
  }
)
nn_get_parameters(modnn3)
modnn3 <- set_hparams( setup(modnn3 ,
                             loss = nn_mse_loss(),
                             optimizer = optim_rmsprop,
                             metrics = list(luz_metric_mae())),
                       input_size = ncol(x))


# Training for ModNN3 Neural Network
# 210sec per epoch
fitted3 <-   fit(modnn3,
                 data = list(x[-testid, ],
                             matrix(y[-testid], ncol = 1)),
                 valid_data = list(x[testid, ],
                                   matrix(y[testid], ncol = 1)),
                 epochs = 100)


matrix(y[-testid], ncol = 1)

list(x[-testid, ],
     matrix(y[-testid], ncol = 1))

plot(fitted3)
summary(fitted3)
fitted3$model
fitted3$ctx



# Predict RMSD values on the test set
test_pred3 <- predict(fitted3, x[testid, ])

test_pred3 <- predict(fitted3, x[testid, ])

plot(y[testid], test_pred3, xlab = "Actual RMSD (Angstrom)",
ylab = "Predicted RMSD (Angstrom)", main = "NN3: 3 Layer
RELU/ELU/CELU")
abline(a = 0, b = 1, col = "purple")

# NN3 MAE calc
length(testid)
```

```
mae_nn3 <- mean(abs(test_pred3 - y[testid]))
mae_nn3



# Addition of Binary Step (Threshold) Hidden Layer
# Threshold of 15 Angstroms
# Intended to ebtter predict RMSD hta appears to be steeped at
the 15 Angstrom level
modnn4 <- nn_module(
  initialize = function(input_size) {
    self$hidden1 <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout1 <- nn_dropout(0.4)
    self$hidden2 <- nn_linear(50, 50)
    self$activation2 <- nn_elu()
    self$dropout2 <- nn_dropout(0.4)
    self$hidden3 <- nn_linear(50, 50)
    self$activation3 <- nn_relu()
    #nn_threshold ( value = 15, inplace = FALSE, threshold = 15)
    self$dropout3 <- nn_dropout(0.4)
    self$hidden4 <- nn_linear(50, 50)
    self$activation4 <- nn_elu()
    self$dropout4 <- nn_dropout(0.4)
    # self$hidden3 <- nn_linear()

    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation1() %>%
      self$dropout1() %>%
      self$hidden2() %>%
      self$activation2() %>%
      self$dropout2() %>%
      self$hidden3() %>%
      self$activation3() %>%
      self$dropout3() %>%
      self$hidden4() %>%
      self$activation4() %>%
      self$dropout4() %>%
      self$output()
  }
```

```
)

modnn4 <- set_hparams( setup(modnn4 ,
                             loss = nn_mse_loss(),
                             optimizer = optim_rmsprop,
                             metrics = list(luz_metric_mae())),
                       input_size = ncol(x))


# Training
# 215 seconds per epoch
fitted4 <-   fit(modnn4,
                 data = list(x[-testid, ],
                             matrix(y[-testid], ncol = 1)),
                 valid_data = list(x[testid, ],
                                   matrix(y[testid], ncol = 1)),
                 epochs = 200)

plot(fitted4)
summary(fitted4)

xfitted4$ctx

# Predict RMSD values for test set
test_pred4 <- predict(fitted4, x[testid, ])

test_pred4 <- predict(fitted4, x[testid, ])
plot(y[testid], test_pred4, xlab = "Actual RMSD (Angstrom)",
ylab = "Predicted RMSD (Angstrom)", main = "3 Layer RELU/ELU
Nueral Network")
abline(a = 0, b = 1, col = "purple")




# Stack on successive RELU ELU hidden layers
modnn5 <- nn_module(
  initialize = function(input_size) {
    self$hidden1 <- nn_linear(input_size, 50)
    self$activation1 <- nn_relu()
    self$dropout1 <- nn_dropout(0.5)

    self$hidden2 <- nn_linear(50, 50)
    self$activation2 <- nn_elu()
    self$dropout2 <- nn_dropout(0.5)
    self$hidden3 <- nn_linear(50, 50)
    self$activation3 <- nn_relu()
    self$dropout3 <- nn_dropout(0.5)

    self$hidden4 <- nn_linear(50, 50)
    self$activation4 <- nn_elu()
    self$dropout4 <- nn_dropout(0.5)

    self$hidden5 <- nn_linear(50, 50)
    self$activation5 <- nn_relu()
    self$dropout5 <- nn_dropout(0.5)

    self$hidden6 <- nn_linear(50, 50)
    self$activation6 <- nn_relu()
    self$dropout6 <- nn_dropout(0.5)

    self$hidden7 <- nn_linear(50, 50)
    self$activation7 <- nn_elu()
    self$dropout7 <- nn_dropout(0.25)

    self$hidden8 <- nn_linear(50, 50)
    self$activation8 <- nn_elu()
    self$dropout8 <- nn_dropout(0.5)

    self$hidden9 <- nn_linear(50, 50)
    self$activation9 <- nn_elu()
    self$dropout9 <- nn_dropout(0.5)

    self$hidden10 <- nn_linear(50, 50)
    self$activation10 <- nn_relu()
    self$dropout10 <- nn_dropout(0.5)

    self$hidden11 <- nn_linear(50, 50)
    self$activation11 <- nn_relu()
    self$dropout11 <- nn_dropout(0.5)

    self$hidden12 <- nn_linear(50, 50)
    self$activation12 <- nn_relu()
    self$dropout12 <- nn_dropout(0.5)

    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden1() %>%
      self$activation1() %>%
      self$dropout1() %>%
  }




      self$hidden2() %>%
      self$activation2() %>%
      self$dropout2() %>%

      self$hidden3() %>%
      self$activation3() %>%
      self$dropout3() %>%

      self$hidden4() %>%
      self$activation4() %>%
      self$dropout4() %>%

      self$hidden5() %>%
      self$activation5() %>%
      self$dropout5() %>%

      self$hidden6() %>%
      self$activation6() %>%
      self$dropout6() %>%

      self$hidden7() %>%
      self$activation7() %>%
      self$dropout7() %>%

      self$hidden8() %>%
      self$activation8() %>%
      self$dropout8() %>%

      self$hidden9() %>%
      self$activation9() %>%
      self$dropout9() %>%

      self$hidden10() %>%
      self$activation10() %>%
      self$dropout10() %>%

      self$hidden11() %>%
      self$activation11() %>%
      self$dropout11() %>%

      self$hidden12() %>%
      self$activation12() %>%
      self$dropout12() %>%

      self$output()
  }
```

```
)

modnn5 <- set_hparams( setup(modnn5 ,
                             loss = nn_mse_loss(),
                             optimizer = optim_rmsprop,
                             metrics = list(luz_metric_mae())),
                       input_size = ncol(x))


#training
#210sec per epoch
fitted5 <-   fit(modnn5,
                 data = list(x[-testid, ],
                             matrix(y[-testid], ncol = 1)),
                 valid_data = list(x[testid, ],
                                   matrix(y[testid], ncol = 1)),
                 epochs = 200)

plot(fitted5)
summary(fitted5)

xfitted2$ctx

# Predict RMSD values for test set
test_pred3 <- predict(fitted3, x[testid, ])

test_pred3 <- predict(fitted3, x[testid, ])
plot(y[testid], test_pred3, xlab = "Actual RMSD (Angstrom)",
ylab = "Predicted RMSD (Angstrom)")
abline(a = 0, b = 1, col = "purple")



# Separation of features Nueral Network
# Attempt to isloate Polar surface areas which appear to be
important for RMSD prediction, likely a function of van der
Waals force
modnn6 <- nn_module(
  initialize = function(input_size) {
    self$hidden1 <- nn_linear(input_size, 50)

    # Use different activation functions for each feature
    self$activation1 <- nn_relu()  # F1, F2, F5, F6, F7, F9
    self$activation2 <- nn_linear()  # F4 F3 (Polar Surface
Areas)
    self$activation3 <- nn_linear()  # F8
    self$dropout1 <- nn_dropout(0.4)
    self$hidden2 <- nn_linear(50, 50)
    self$dropout2 <- nn_dropout(0.4)

    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    # Split the input tensor based on the feature indices
    x1 <- x[, 1:7]  # F1, F2, F3, F4, F5, F6, F7
    x2 <- x[, 8]  # F8
    x3 <- x[, 9]  # F9

    # Forward pass for each feature with the corresponding 
activation function
    out1 <- x1 %>%
      self$hidden1() %>%
      self$activation1() %>%
      self$dropout1() %>%
      self$hidden2() %>%
      self$activation2() %>%
      self$dropout2()

    out2 <- x2 %>%
      self$hidden1() %>%
      self$activation3()  # Use linear activation function for 
F8

    out3 <- x3 %>%
      self$hidden1() %>%
      self$activation1()  # Use ReLU activation function for F9

    # Concatenate the output tensors from each feature and pass 
through the output layer
    torch$cat(list(out1, out2, out3), dim = 2) %>%
      self$output()
  }
)

modnn6_setup <- setup(modnn6 ,
                      loss = nn_mse_loss(),
                      optimizer = optim_rmsprop,
                      metrics = list(luz_metric_mae()))

modnn6 <- set_hparams(modnn6_setup, input_size = ncol(x))

# Training ModNN6 Separation of Features Nueral Network
# Approximately 240 seconds for each epoch
fitted6 <- fit(modnn6,
               data = list(x[-testid, ],
                           matrix(y[-testid], ncol = 1)),
               valid_data = list(x[testid, ],
                                 matrix(y[testid], ncol = 1)),
               epochs = 100)
```

```
# Group Method of Data Handling Neural Network (GMDH)
# Define the base models for the GMDH NN

# GMDH function is from GMDHreg library
# Training Multilayered Iterative (MIA) GMDH Neural Network

# Fit GMDH NN model
modnn7 <- gmdh.mia(X = as.matrix(protein[-testid, 2:10]),
                      y = as.matrix(protein[-testid, 1]),
                      prune = ncol(protein[-testid, 2:10]),
                      criteria = "PRESS")
# PRESS criteria: Predicted Residual Error Sum of Squares takes
into account all information in data sample and it is computed
without recalculation each test point.
# "test" is alternative argument, estimation of RMSE and is
computationally more efficient

summary(modnn7)

# Predict on validation set for GMDH NN
pred.modnn7 <- predict(modnn7, as.matrix(protein[testid, 2:10]))

sum(is.na(pred.modnn7[,1]))
# For set.seed(13), there are 27 NA values

# Get row numbers for NA vlaues
missing_rows <- which(is.na(pred.modnn7[,1]))
missing_rows

pred.modnn7_clean <- pred.modnn7[-missing_rows, ]
# Should be 0
sum(is.na(pred.modnn7_clean))

# Remove missing rows from the test set
```

```
protein.test <- protein[testid,][-missing_rows,]
# Calculate MAE on test set
mae <- mean(abs(pred.modnn7_clean - protein.test[,1]))
# With set.seed(13), MAE is 4.359865
mae

# Plot predicted RMSD against actual RMSD
plot(protein.test[,1], pred.modnn7_clean, xlab = "Actual RMSD
(Angstrom)", ylab = "Predicted RMSD (Angstrom)", main = "NN7:
136 Layer MIA GMDH NN")
abline(a = 0, b = 1, col = "purple")




# Fit GMDH GIA NN model
modnn8 <- gmdh.gia(X = as.matrix(protein[-testid, 2:10]),
                   y = as.matrix(protein[-testid, 1]),
                   prune = 9*2,
                   criteria = "test")

ncol(protein[-testid, 2:10])
# Predict on validation set for GMDH NN
pred.modnn8 <- predict(modnn8, as.matrix(protein[testid, 2:10]))

sum(is.na(pred.modnn8[,1]))
# For set.seed(13), there are 27 NA values

# Get row numbers for NA vlaues
missing_rows <- which(is.na(pred.modnn7[,1]))
missing_rows

pred.modnn7_clean <- pred.modnn7[-missing_rows, ]
# Should be 0
sum(is.na(pred.modnn7_clean))

# Remove missing rows from the test set
protein.test <- protein[testid,][-missing_rows,]
# Calculate MAE on test set
mae <- mean(abs(pred.modnn7_clean - protein.test[,1]))
# With set.seed(13), MAE is 4.36
mae
```