NOVOZYMES ENZYME STABILITY PREDICTION

A Kaggle Competition

Amudha Giridharan, Chaitrashree Sadasivappa, Paul Merica,
Sai Shiva Bhaskar Emani, Soundarya Alwa

DSA 6000 - Group 10 – Final Project

NOVOZYMES ENZYME STABILITY PREDICTION

Table of Contents

1. Abstract

This project report details the work done by the team to predict the enzyme stability for the Kaggle competition hosted by Novozymes. The competition background and business case are briefly discussed in this report, followed by the initial exploratory data analysis done by the team to understand the training and test data sets. The report then discusses the various feature engineering approaches adopted and the external sources used to enhance the test and training data sets. The various machine learning models experimented including the Xgboost cross validation and deletion ensemble, and the results achieved are elaborated.

The report then summarizes the challenges faced by the team in terms of bioinformatics subject matter expertise and the challenges in adopting R instead of Python, which is the most widely used machine language for this specific scenario. Finally, the achievements of the team as of date, including publishing the proof-of-concept notebook in Kaggle discussion and getting into top 50% of the leaderboard are underlined, followed by the plans for model improvement as the competition is still on.

2. Project Background

1. Business Case

Enzymes are widely used in many industries. They are heavily used in laundry and dishwashing detergents where they remove stains and enable low temperature washing and concentrated detergents. Other enzymes improve the quality of bread, beer, and wine, or increase the nutritional value of animal feed. Enzymes are also used in the production of biofuels where they turn starch or cellulose from biomass into sugars which can be fermented to ethanol.

However, many enzymes are only marginally stable, which limits their performance under harsh application conditions. Understanding and accurately predicting enzyme protein stability is a fundamental problem in biotechnology. Its applications include enzyme engineering for addressing the world's challenges in sustainability, carbon neutrality and more. Improvements to enzyme stability could lower costs and increase the speed scientists can iterate on concepts.

2. Competition Goal

The goal of this competition hosted by Novozymes is to predict the thermostability of enzyme variants. The experimentally measured thermostability (melting temperature) data includes natural sequences, as well as engineered sequences with single or multiple mutations upon the natural sequences. The competitors are expected to develop a model to predict/rank the thermostability of enzyme variants based on experimental melting temperature data, which is obtained from Novozymes's high throughput screening lab.

3. Data Sets details

1. Train data

The train data provided by competition host contains ~31K protein sequences(enzymes) with mutations upon the natural sequences (wildtypes), pH at experiment and the respective melting temperature, collected from multiple data sources. To our observation, the train data contained single point substitution mutations only. There were no observations which were either deletion mutations or wildtype itself.

Screenshots of train data (7.3.1) and example of single point substitution mutation (7.3.2) provided in Appendix.

2. Test data

The test data provided by competition host contained 2,413 mutant protein sequences derived from single wild type, of which 77 were deletion mutations of this wild type and 2,335 were substitution mutations of the same given test wild type. The 3-dimensional structure of the test enzyme's wildtype as predicted by AlphaFold, was also provided.

Screenshots of test data (7.3.3) and graphical representation of wild type structure (7.3.4) provided in Appendix.

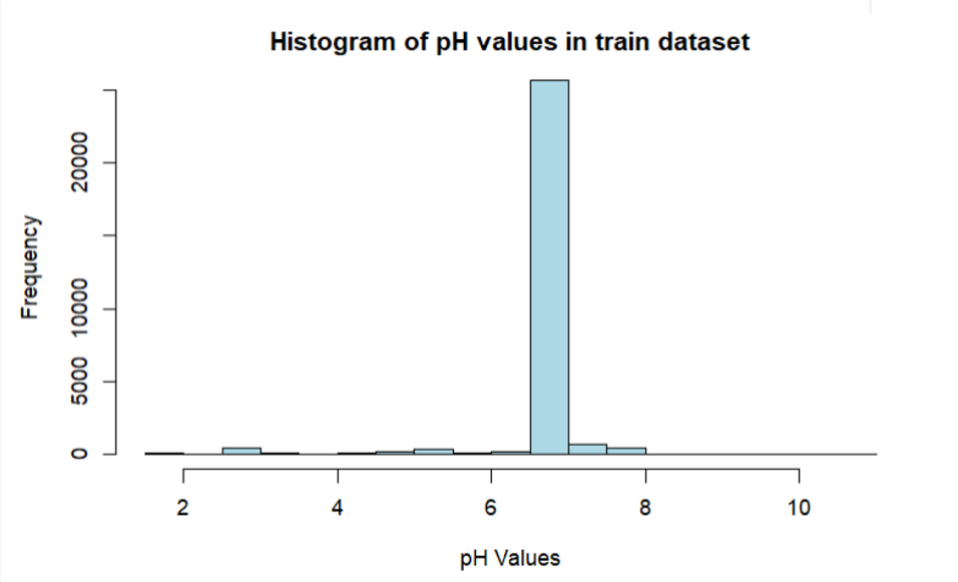
3. External sources

External data sources were allowed in this competition: We used related data from microbiologist Jin Yuan (https://github.com/JinyuanSun/mutation-stability-data) for model build experiments (only for dTM and normalized dTm models). However, this data wasn't part of the best final model submitted by the team.

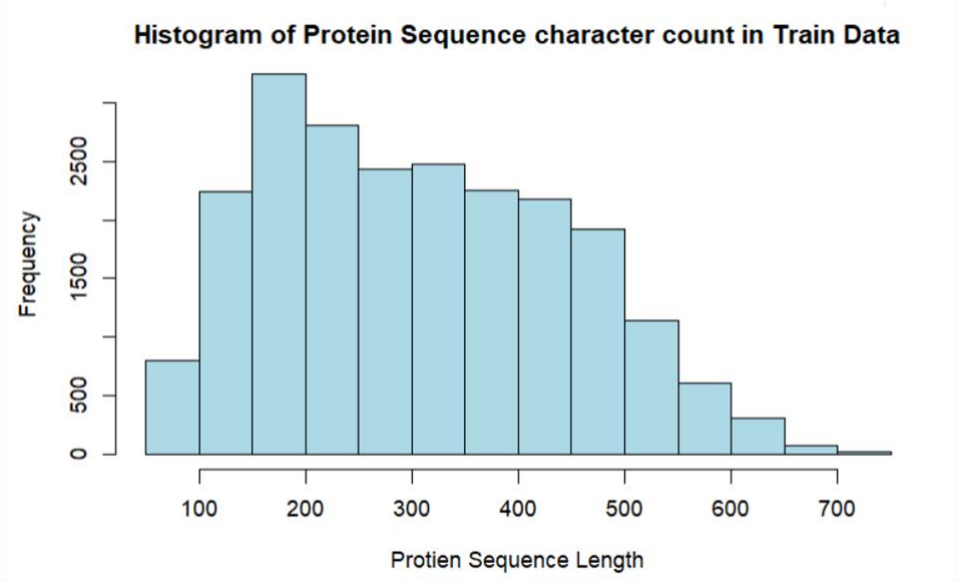
4. Exploratory Data Analysis

We performed EDA to understand the data to devise the feature extraction and modeling methods accordingly.

1. Data Analysis and Observations



We observed that pH values for train data have a mean of 6.8 and a median of 7. All the records in test data had pH value of 8. Different pH values affect thermal stability in different ways. So, our model should be trained to learn the relationship between pH and thermal stability of a protein.

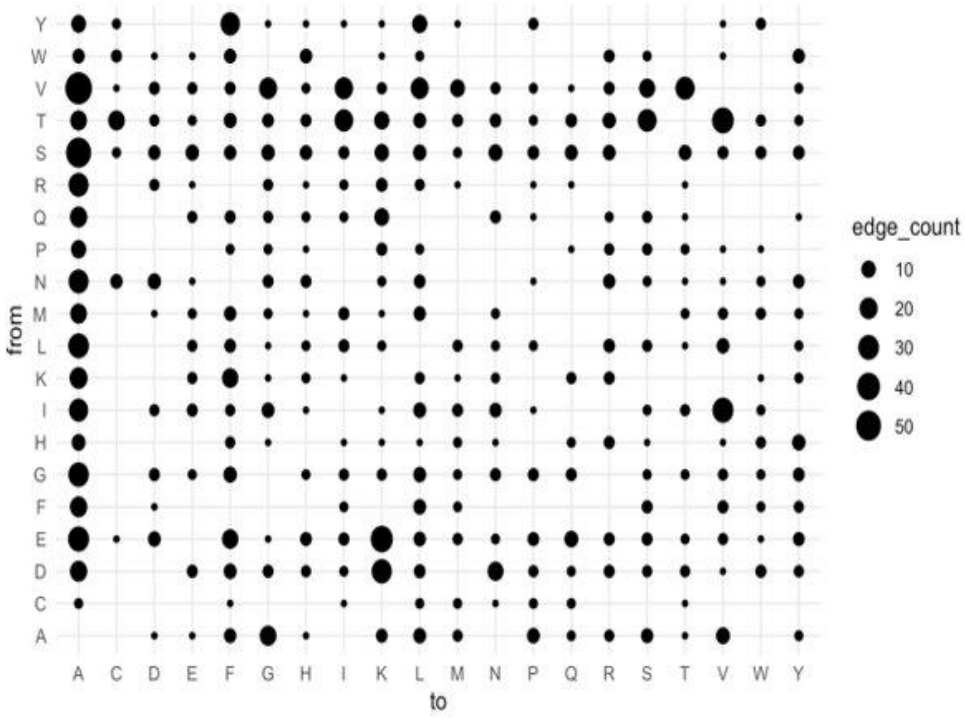


We grouped train data based on the protein sequences character count and found that train data had protein sequences with character count varying in wide range between 5 to 32000. But few of the character counts were represented by very few proteins only. So, we filtered train data to get only those character counts which were having representation from atleast 20 protein sequences and above.

After this filtering, train data had a protein sequence length from a range of 53 to 747, while 97% of the test data protein sequence length consists of 221 and the remaining 3% of 220. This may make it difficult to train the model due to variance in the sequence lengths between train and test data.

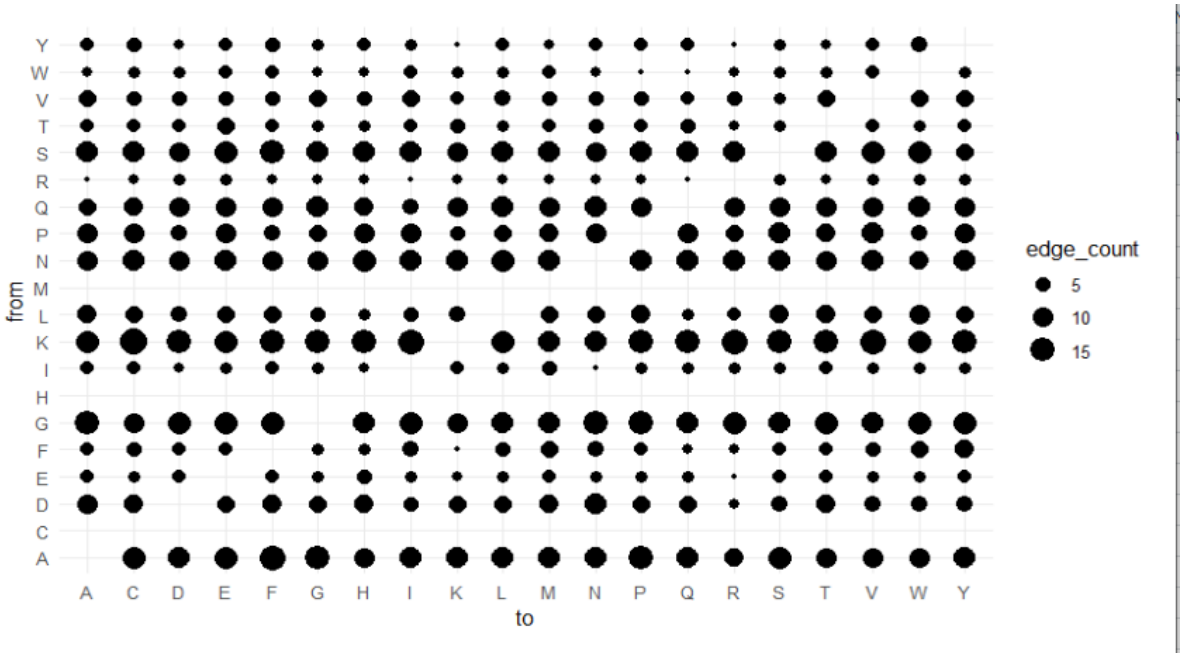
Analyzing further on the mutations, we plotted node graph for train and test data to identify the from and to residue values.

Node Graph for Train Data



While plotting a Node graph for Train data we observe that most of the mutations are happening from all the variables to “A” compared to all the other mutations which is very different from the test data.

Node Graph for Test Data



While plotting a node graph for the Test data we observe that there are few mutations that happen to residues “C”, “H”, “M” from other residues, but no vice-versa mutations were found. Also, the mutations seem to uniform across residues when compared to train data. This might be the case as the test set is only based on a single wild type, while the train set has multiple wild types.

2. Data related challenges

Based on the EDA, we were able to list out the data related challenges as below:

1. **Train data deficiency**
    a. Train data had single point substitution mutations only
    b. Test data had both single point substitution and deletion mutations
    c. The model might not be able to predict tm for deletion mutation as there was no training data available for this scenario
2. **Training data correctness**
    a. Train data was a compilation from multiple data sources
    b. Similar protein sequences at the same pH had different tm, based on the data source in the train data
    c. Test data was from a single data source, and wasn't referenced in the training data source
    d. The model might not be able to predict the tm correctly for the new data source
3. **Test data singularity**
    a. Entire test data set was based on single wild type
    b. Train data did not contain granular level observations to match with test set

5. Approach

1. Feature engineering

1. Proof of concept

The team performed POC for below challenges identified in section 4.2. The technical challenges can be further detailed as below:

i. How to explore this solution execution in R instead of Python which many Kagglers had adopted
ii. How to convert the protein sequence (list of characters) to numerical values which can be used to predict test protein sequence's thermal stability (Tm).
iii. How to identify similarities between the protein sequences to be able to group them to find related sequences (potential mutations)
iv. How to find the wild type mapping for the given train protein sequences. This information is critical given that the test data has one wild type, and all other observation are its mutations.

The POC notebook with the Bioseq package functions for feature engineering was published by our team in Kaggle. Link is https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction/discussion/371747

The feature engineering outcomes from POC are detailed from 5.1.2 to 5.1.4.

2. Amino acid weightage within a protein sequence

The team did a POC to find if bioseq R package can help with creating amino acid weightage for the sequences, and the results were affirmative. The proportion of each amino acid was calculated for all the sequences using Seq_stat_prop function Bioseq package from R With this weightage calculated, we can tie back the change in tm to the variation in weightage between mutations

Example code and results are added in appendix (7.3.5)

3. Protein sequence clustering

Again, we did POC for methods to identify the potential mutations in training data, and we identified that seq_nchar and seq_cluster function from bioseq R package can help with this. We computed the aminoacid count in each of the sequence using seq_nchar function. This helped us to group potential mutations together. Substitution mutations would have same character size as of the wild type and deletion mutation will have one-character lesser size. Example code and results are added in appendix (7.3.6)

4. Protein sequence wild type identification

We then used seq_cluster function to compute consensus and assign representative numbers for clusters. With the clustered protein sequences, we used the seq_consensus to find the wild type. The POC we did on various data sets from test and train data confirmed the reliability of this function. For each amino acid character count and cluster combination, we used the consensus technique to identify the wild type, only if the total mutation in that category is substantial. Trivial clusters with protein sequences less than the threshold (set to 20) were filtered out. We now had the wild type mapped to each protein sequence for our further analysis

Example code and results are added in appendix (7.3.7)

5. Mutation position and type

With the wild type identified, we then added the information of residue position of mutation and type of mutation (substitution/deletion/wildtype) and appended that to the training set. This information further helped to enhance the model accuracy

Example code and results are added in appendix (7.3.8)

2. Modelling techniques

The data provided by Kaggle competition host was very sparse and not necessarily well documented, as discussed above. Because of these factors, we decided that decision tree models like XGBoost, Random Forest, TREE, CTree, Rpart, and GBM would be the best ways to model using this data. We believed these models could create branches to isolate the effects of mutations on the protein sequences.

1. Target variable determination

This approach followed was unique as we experimented with the model on 3 different targets.

Since the objective of the competition is to rank the thermal stability for point mutations to a wildtype string, there were three options identified.

a. To directly predict the thermal stability (Tm) of each mutation sequence of the original wildtype string by using the weightage and pH and rank the observations based on the predicted Tm
b. model the change in Tm (dTm) due to the mutation. This would allow us to rank the test data set by highest to lowest Tm in the same way, because we know that the test data set are mutations of one wildtype sequence.

c. Model the change in the free energy of the protein (Gibbs free energy), due to the mutation (ddG). ddG is proved to be directly proportional to dTm and should give similar results as in the above two approaches.

Modelling on dTm also has the additional advantage of being able to model on more data. Since the Jin data mentioned above is formatted in dTm change and ddG mutation with the mutation named (Appendix Jin). Just appending the dTm data to our reformed dTm data gives us about 2632 rows of data to model on which is useful versus the 1634 rows we use to model on Tm solely.

Finding a way to append the ddG data would allow us to use additional data (6642 rows) So, our next step was normalizing the Jin ddG data to make it comparable to dTm. Normalization in this context is scaling the targer data (ddG and dTm) to have a mean of 0 with a standard deviation of 1. We did this for every "grouping", or in other words every wildtype group with its mutation variants. This hopefully would allow the models to be able to see how the mutation and the position changes the thermal stability of a protein sequence.

2. Model Analysis

We then compared the effectiveness of the 6 modelling methods listed above (XGBoost, RF, etc.) on our three targets mentioned by using Mean Squared Error (MSE). This is essentially a measure that shows how much the model missed. We calculated this by splitting each individual data set into training and testing data sets. Then testing each of the models on the validation set to get the Mean Squared Error to compare how each of the models performed.

3. Hyper parameter tuning

We then started the process of tuning hyperparameters of the best performing models in each of the target sequences. This is discussed more below in the "Model Performance Summary". We decided to tune them and train the model on the full training data since the test data is different from the training data.

4. Deletion Ensemble

The deletion ensemble approach was the resultant decision, while we analyzed on the methodology to handle deletions. The training data has no deletions to predict the deletion scenarios in test set. So we ranked the deletion mutations using the "b factor" from the test data and the blosum data matrix alone (average rankings from b factor + blosum matrix), while the substitution mutations were ranking included the model's predictions as well (average rankings from the model + b factor + blosum matrix). We found this approach helped improved our model accuracy.

6. Results

1. Model performance summary

We ran 6 models for each of the targets and got the following results with our training and validation set.

Validation Set MSE	XGB	Random Forest	CTREE	RPART	TREE	GBM
Tm Prediction	33.9	38.7	41.9	45	44.8	38.9
dTm Prediction	20.5	23.7	23	23.4	23.3	24.36
Normalized Prediction	.83	.79	.92	.92	1.02	.92

From the above table we can observe that

a. XGBoost is the best model for Tm prediction (MSE of 33.9) and dTm prediction (MSE of 20.5).
b. Random Forest (MSE of .79) is the best model for the Normalized Prediction model.

From this point on we did hyper parameter tuning on these models to make them as accurate as possible. This was done through a method called Cross-Validation. Cross-Validation is a method that prevents overfitting by using different combinations of the training and testing (in our case validation set) to train the model.

We ended up finding that in the Tm Model the optimal amount of "nrounds" (which is the number of decision trees in the final model) is 16 and is 28 in the dTm model. We also tuned the amount of mtry in the Random Forest model (the number. of variables to randomly sample as candidates at each split) which ended up being 18. This was all to make sure we had the optimal parameters before submitting to Kaggle.

We then averaged our rankings out with the b-factor rankings and the Blosum rankings then got the following scores from Kaggle with these models.

2. Final model choice

As we can observe, our best model was the ensemble Tm model. We believe this is mainly because we a wide enough breadth of information for the dTm models to accurately determine the change in thermal stability of mutants when compared to wildtype's thermal stability. We also felt that if we incorporated CIF and ESM data into our models, it would provide improved performance.

7. Summary

1. Challenges

1. Good amount of bioinformatics knowledge was needed to understand terminologies and correlations required to attempt this competition. The team had to do lot of research on bioinformatics to get up-to speed with other kagglers who started before us.
2. There was limited information in the given train data and test data. As training data alone wasn't sufficient to predict test observations tm we used some data from external sources as the competition encourages external data utilization.
3. As we had short time period to finish this, we couldn't incorporate external PDB files for each protein sequence into our models as this required additional deep dive analysis to know how to use the content from those files (most submissions from other kagglers have been working on this for multiple months).
4. Also, we were searched for any pretrained protein language model in R as these models help to extract features from protein sequences. But we didn't have much help from R for this part. So, we had to try some other approaches to get features to train our model. Other kagglers were using its counterpart for the same in python is, ESM package from Facebook which gave them a good starting point.
5. And this dataset requires complex feature engineering for accurate prediction

2. Achievements

1. Most of the kagglers are attempting this competition with python so there were not many POCs done for using R for this competition on Kaggle. We attempted this competition with R and did a POC using R bioseq package and were successful in getting remarkable outcome. We have uploaded this POC on Kaggle to help others who want to try this with R and link to
   Kaggle notebook is given below.

https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction/discussion/371747

2. As the given train dataset didn't suffice our modeling requirements, we explored external datasets which could be combined with our competition data. Based on our research we selected data from external sources and were successful in combining them with our train data to get new features to use for model training. Also, we leveraged Blosum matrix and the b-factor from PDB file to support model performance.

3. With all these steps we were able to achieve a good ranking (our team is ranked in top half) on this Kaggle competition in short duration.

3. Model improvement plans

We have plans to improve our model performance by doing additional feature engineering using relevant data from external sources, using different models, using cross validation techniques to reduce overfitting. We are working on using the external PDB files for each protein sequence to get new useful features which would help the model train better on the given train dataset.

PDB and CIF files contain of a protein contains several properties like unit cell values, atom names and their coordinates and any structural model quality indicators,

Few feature engineering ideas are:

1. To calculate the delta values for all the properties of a wildtype with mutated element and using this as a new feature in our train dataset. These new delta values let the model learn the difference between wildtype and Mutant sequence.
2. To use the x, y, z co-ordinates of the corresponding wildtype and mutated element in each sequence to calculate the location id as a new feature and to use this feature as well in training our model.

4. Lessons learnt & Recommendations

1. We learned that to truly be a good data scientist you need to be fluent in multiple statistical coding languages. Most of the analysis done in Kaggle is in Python, so being able to understand Python and incorporate the logic from that code that in R, Java or your preferred language is extremely valuable

2. Predicting thermal stability is very hard and there's a reason there is a competition for trying to fix it.

3. Making Statistical models based on complex topics requires a lot of research into the underlying facts surrounding the topic at hand. We ended up watching a lot of biology videos and reading posts to truly understand how enzymes worked to help grasp the assignment. This is very similar to how models work in business cases, as a lot of times being able to describe the underlying business function of a model is just as important (if not more important) than the accuracy of the model.

4. We learned how to use new R packages like bioseq for protein sequence handling along with exposure to data wrangling packages like tidyverse that allow us to manipulate data more easily.

5. A genuinely good model takes a lot of time. We still have so many avenues we could have explored but we only had a couple of months, where someone could spend years doing a project like this. Modelling protein distances, placement of molecules, and the millions of other things surrounding mutations.

8. Appendix

1. Terminologies

Terminologies	Definition
Enzymes	Enzymes are proteins that act as catalysts in the chemical reactions of living organisms
Amino Acids	Amino acids are molecules that combine to form proteins. Example: histidine, leucine etc
Wildtype	A term used to describe a enzyme (in this competition context) when it is found in its natural, non-mutated (unchanged) form
Mutant	Protein with an amino acid change/mutation from its wildtype (in the context of this competition)
tm	The protein melting point (TM) is defined as the temperature at which the protein denatures.
dtm	Delta melting temperatures between two enzymes
Denaturation	Process in which a molecular structure deviates from its original state when exposed to a high temperature, chemical reactions etc.
Spearman's correlation	A nonparametric measure of rank correlation (statistical dependence between the rankings of two variables). It assesses how well the relationship between two variables can be described using a monotonic function.
PDB	Protein Data Bank. The PDB format provides for description and annotation of protein and nucleic acid structures including atomic coordinates, secondary structure assignments, as well as atomic connectivity.
ddg	The change in energy between the folded and unfolded states (ΔG folding) when a point mutation is present.
blosum	BLOSUM (BLOcks SUbstitution Matrix) matrix is a substitution matrix used for sequence alignment of proteins
b factor	B-factor aka Debye–Waller factor, temperature factor, or atomic displacement parameter, is used in protein crystallography to describe the attenuation of X-ray or neutron scattering caused by thermal motion.

2. GitHub Link for code and Knit file

Git Hub Link for our Project code

R markdown with run results

3. Screenshots & Samples

1. Train data snapshot

seq_id	protein_sequence	pH	data_source	tm
0	AAAAKAAALALLGEAPEVVDIWLPAGWRQPFRVFRLERKGDGVL...	7	doi.org/10.1038/s41592-020-0801-4	75.7
1	AAADGEPLHNEEERAGAGQVGRSLPQESEEQRTGSRPRRRRDLG...	7	doi.org/10.1038/s41592-020-0801-4	50.5
2	AAAFSTPRATSYRILSSAGSGSTRADAPQVRRLHTTRDLLAKDYYA...	7	doi.org/10.1038/s41592-020-0801-4	40.5
3	AAASGLRTAIPAQPLRHLLQPAPRPCLRPFGLLSVRAGSARRSGLL...	7	doi.org/10.1038/s41592-020-0801-4	47.2
4	AAATKSGPRRQSQGASVRTFTPFYFLVEPVDTLSVRGSSVILNCSA...	7	doi.org/10.1038/s41592-020-0801-4	49.5
5	AACFWRRTVIPKPPFRGISTTSARSTVMPAWVIDKYGKNEVLRFT...	7	doi.org/10.1038/s41592-020-0801-4	48.4
6	AACFWRRTVIPKPPFRGISTTSARSTVMPAWVIDKYGKNEVLRFT...	7	doi.org/10.1038/s41592-020-0801-4	45.7

2. Train data - single point mutation random example

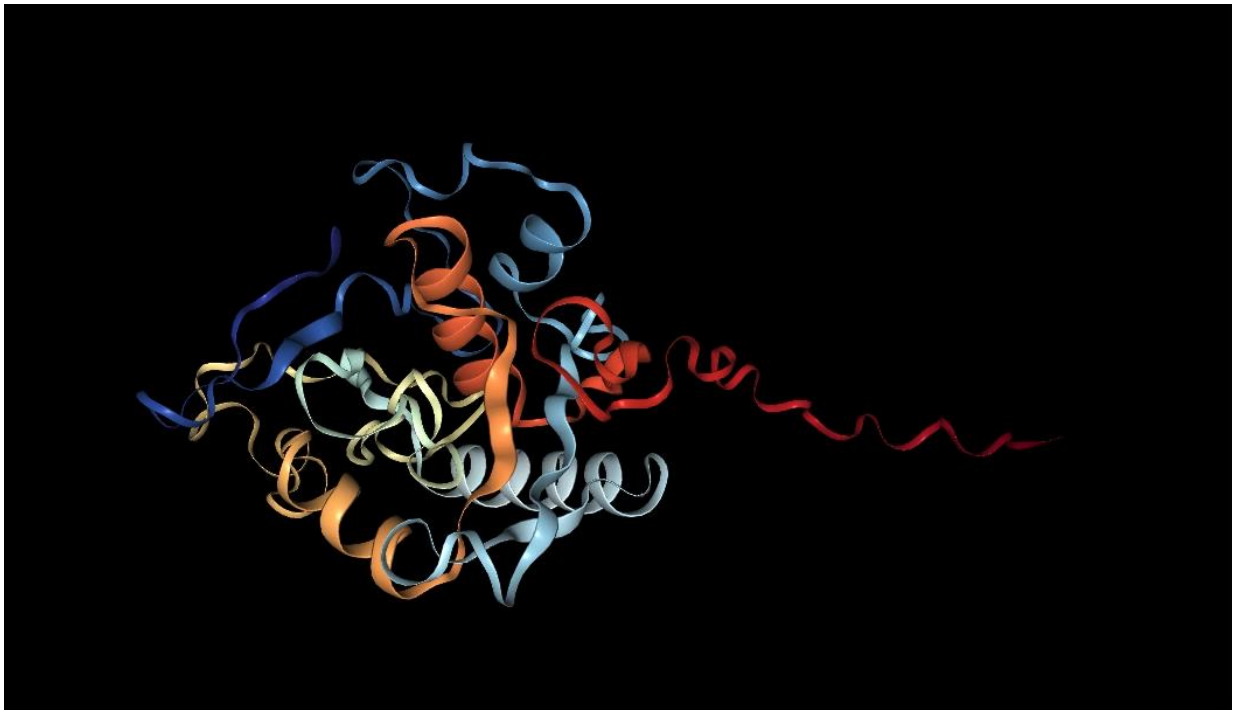
Wildtype	**MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA**
Examples of Single point mutation	AKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA
	MKGASKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA
	MKGMAKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA
	MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKAEGRIGA
	MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGAIGA
	MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIAA

Note: Highlighted in red are the single point mutated amino acids in the mutant protein sequences, when compared to the wild type protein sequence (source)

3. Test data – snapshot

seq_id	protein_se...	pH	data_source
31390	VPVNPEPDATSVENV AEKTGSGDSQSDPIK ADLEVKGQSALPFDV DCWAILCKGAPNVLQ RVNEKTKNSNRDRSG ANKGPFKDPQKWGIK ALPPKNPSWS...	8	Novozymes
31391	VPVNPEPDATSVENV AKKTGSGDSQSDPIK ADLEVKGQSALPFDV DCWAILCKGAPNVLQ RVNEKTKNSNRDRSG ANKGPFKDPQKWGIK ALPPKNPSWS...	8	Novozymes
31392	VPVNPEPDATSVENV AKTGSGDSQSDPIKA DLEVKGQSALPFDVD CWAILCKGAPNVLQR VNEKTKNSNRDRSGA NKGPFKDPQKWGIKA LPPKNPSWSA...	8	Novozymes

4. Test wild type sequence structure graph



5. Amino acid weightage for a protein sequence

Below screenshot shows a sample protein sequence and its corresponding numeric weightages

```
> seq_stat_prop(aa('MNQSVSSLPEKDIQYQLHPYTNARLHQELGPLIIERGEGIYVYDDQGKGYIEAMAGLWSAALGFSNQRLIKAAEQQFNTLPFYHLFSHKSHRPSIELAEKLIAMAPVPMSKVFFTN
SGSEANDTVVKMVWYLNNALGKPAKKKFISRVNGYHGITVASASLTGLPGNQRGFDLPLPGFLHVGCPHHYRFALAGESEEHFADRLAVELEQKILAEGPETIAAFIGEPLMGAGGVIVPPRTYWEKIQKVCRKYD
ILVIADEVICGFGRTGQMFGSQTFGIQPDIMVLSKQLSSSYQPIAAILINAPVFEGIADQSQALGALGHGFTGSGHPVATAVALENLKIIEEEESLVEHAAQMGQLLRSGLQHFIDHPLVGEIRGCGLIAAVELVGD
RVSKAPYQALGTLGRYMAGRAQEHGMITRAMGDAVAFCPPLIVNEQEVGMIVERFARALDDTTQWVG
+ '))
[[1]]
          A           C           G           T           W           S           M           K           R           Y
0.101098901 0.010989011 0.101098901 0.035164835 0.008791209 0.054945055 0.028571429 0.039560440 0.041758242 0.030769231
          B           D           H           V           N           E           F           I           L           P
0.000000000 0.032967033 0.035164835 0.065934066 0.028571429 0.068131868 0.043956044 0.070329670 0.094505495 0.052747253
          Q
0.054945055
```

Below screenshot shows the weightage data merged into training data set.

	A	C	G	T	W	S	M	K	R	Y
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.1010989	0.010989011	0.1010989	0.03516484	0.008791209	0.05494505	0.02857143	0.03956044	0.04175824	0.03076923
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.0989011	0.010989011	0.1010989	0.03516484	0.008791209	0.05494505	0.02857143	0.04175824	0.04175824	0.03076923
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.0989011	0.010989011	0.1010989	0.03516484	0.008791209	0.05494505	0.02637363	0.03956044	0.04175824	0.03076923
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.0989011	0.010989011	0.1010989	0.03516484	0.008791209	0.05494505	0.02637363	0.03956044	0.04175824	0.03296703
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.0989011	0.010989011	0.1010989	0.03516484	0.008791209	0.05274725	0.02857143	0.03956044	0.04175824	0.03076923
IQYQLHPYTNARLHQELGPLIIERGEGIYVY...	0.0989011	0.010989011	0.1010989	0.03516484	0.008791209	0.05494505	0.02857143	0.03956044	0.04175824	0.03076923

6. Train Protein sequence clustering and Wild type identification

The below is representative sample of amino acid character count and cluster sequencing.

Filter

e	pH	data_source	tm	charcnt	clusternum	wildtype	A	C
RWPREVLDLVRKVAEENGRSVNSEIYQRV...	7.5	10.1038/nsb0894-518	58.0	53	1		0.05660377	0.0000000
TIVSDGKPQTDNDTGMISYKDANGNKQ...	7.0	doi.org/10.1038/s41592-020-0801-4	62.9	53	2		0.01886792	0.0000000
RWPREVLDLVRKVAEENGRSVNSEIYQR...	7.5	10.1038/nsb0894-518	58.7	53	1		0.05660377	0.0000000
TTFAIESTVELSFEEAMTPEVIEKYNAKTT...	7.0	doi.org/10.1038/s41592-020-0801-4	37.5	53	3		0.07547170	0.0000000
RWPREVLDLVRKVAEENGRSVNSEIYQR...	7.5	10.1038/nsb0894-518	58.1	53	1		0.05660377	0.0000000
RWPREVLDLVRKVAEENGRSVNSEIYQRV...	7.5	10.1038/nsb0894-518	59.2	53	1		0.05660377	0.0000000
.RWPREVLDLVRKVAEENGRSVNSEIYQR...	7.5	10.1038/nsb0894-518	57.5	53	1		0.05660377	0.0000000
RWPREVLDLVRKVAEENGRSVNSEIYQR...	7.5	10.1038/nsb0894-518	59.6	53	1		0.05660377	0.0000000
RWPREVLDLVRKVAEENGRSVNSEIYQRV...	7.5	10.1038/nsb0894-518	55.5	53	1		0.05660377	0.0000000
.RWPREVLDLVRKVAEENGRSVNSEIYQR...	7.5	10.1038/nsb0894-518	74.1	53	1		0.05660377	0.0000000

Showing 1 to 10 of 22,507 entries, 29 total columns

Below screenshots show the table across character counts and cluster and the matrix values represent the count of protein sequences in that combination.

ident

```
> table(train_filtered$charcnt,train_filtered$clusternum)

     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 1
 53 52  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 63  1  1  1  1  1  1 21  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 66  1  1  1  1  2 20  1  1  1 53  1  1  1  2  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 67  1  1  1  1  1  1 45  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 71  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0
 84  1  1  1  1  1  1  1  1  1  1  1 55  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 85  1  1  1  1  1  4  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0
 87  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0
 89  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  2  1  1  1  1  1  1  1  1  0
 90  1  1  1  1  1  1  1  1  1  1  1 13  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0
```

```
 [1] "AKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MAGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
 [3] "MKAMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGASKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
 [5] "MKGMAKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSAMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
 [7] "MKGMSKAPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMAQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
 [9] "MKGMSKMLQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPAFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[11] "MKGMSKMPQANLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFALRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[13] "MKGMSKMPQFNARWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLAWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[15] "MKGMSKMPQFNLRAPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWAREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[17] "MKGMSKMPQFNLRWPAEVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPRAVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[19] "MKGMSKMPQFNLRWPREALDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVADLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[21] "MKGMSKMPQFNLRWPREVLALVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVLDAVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[23] "MKGMSKMPQFNLRWPREVLDLVAKVAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVLDLVRAVAEENGRSVNSEIYQRVMESFKKEGRIGA"
[25] "MKGMSKMPQFNLRWPREVLDLVRKAAEENGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVLDLVRKVAAENGRSVNSEIYQRVMESFKKEGRIGA"
[27] "MKGMSKMPQFNLRWPREVLDLVRKVAEANGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVLDLVRKVAEEAGRSVNSEIYQRVMESFKKEGRIGA"
[29] "MKGMSKMPQFNLRWPREVLDLVRKVAEEAGRSVNSEIYQRVMESFKKEGRIGA" "MKGMSKMPQFNLRWPREVLDLVRKVAEENARSVNSEIYQRVMESFKKEGRIGA"
```

8. Mutation type and position identification

The below is corresponding position and mutation information from the data in previous screenshots.

```
> head(train_filtered[ , c("protein_sequence","wildtype","charcnt","clusternum" ,"type", "resid","wt","mut")])
                                protein_sequence                                wildtype charcnt clusternum type resid wt mut
1 AKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     1  M   A
2 MAGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     2  K   A
3 MKAMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     3  G   A
4 MKGASKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     4  M   A
5 MKGMAKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     5  S   A
6 MKGMSAMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA MKGMSKMPQFNLRWPREVLDLVRKVAEENGRSVNSEIYQRVMESFKKEGRIGA      53          1  SUB     6  K   A
```

4. Code

```
---
title: "Novozymes Prediction"
output:
  word_document: default
  html_document: default
  pdf_document: default
date: "2022-12-09"
---
```

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

Packages

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r}
library(xgboost)
library(bioseq)
library(dplyr)
library(bio3d)
library(tidyverse)
library(rpart)
library(bsnsing)
library(C50)
library(party)
library(tree)
library(randomForest)
library(xgboost)
library(gbm)
library(ggplot2)
library(patchwork)
library(htmlwidgets)
library(NGLVieweR)
library(stringi)
library(ggraph)
library(tidygraph)
```

Uploading Data

```{r}


# Original Training Data from Kaggle
train = read.csv("train.csv")

# Training data update
train_update= read.csv("train_updates_20220929.csv")

# Test data we will be ranking and turning in
test = read.csv("test.csv")
test_pdb <- read.table("wildtype_structure_prediction_af2.pdb",fill=T)

# PDB file containing our one test wild type
```

```
pdbfile = read.pdb("wildtype_structure_prediction_af2.pdb" )

proteinsequence = pdbfile$atom
# Wild type sequence provided in the Kaggle "Dataset Description":
wtseq <-
'VPVNPEPDATSVENVALKTGSGDSQSDPIKADLEVKGQSALPFDVDCWAILCKGAPNVLQRVNEK
TKNSNRDRSGANKGPFKDPQKWGIKALPPKNPSWSAQDFKSPEEYAFASSLQGGTNAILAPVNLAS
QNSQGGVLNGFYSANKVAQFDPSKPQQTKGTWFQITKFTGAAGPYCKALGSNDKSVCDKNKNIA
GDWGFDPAKWAYQYDEKNNKFNYVGK'

 ## let us remove all the rows with data issues as advised by host
train <- train[!(train$seq_id %in% train_update$seq_id),]
str(train) # 28956 obs

#Classify the train update data and identify the rows to be used
train_update_null <-  subset(train_update,is.na(train_update$tm))
head(train_update_null) # we are going to ignore this dataset. using only for validation purpose
train_update_swap <- subset(train_update,!is.na(train_update$tm))
head(train_update_swap)

#use the train_update_swap dataframe to add back the rows with swapped pH & tm
train <- rbind(train, train_update_swap)
str(train) # 28981 obs

# check for duplicates
train[duplicated(train),] # no duplicates
```

Train Data Wrangling

Step 1: add sequence char count to train data

```{r}

train$charcnt <- seq_nchar(aa(train$protein_sequence))

```

Step 2: Let us filter out the protein sequences which have less frequency of occurrance

```{r}
seq_freq_threshold = 20

train_filtered <- train %>%
  group_by(charcnt) %>%
  filter(n() >= seq_freq_threshold)

```

```{r}
hist(train_filtered$charcnt, col = 'lightblue' ,main = "Histogram of Character Count Frequencies")
```

```

### Step 3: add cluster number to train data

```{r}

train_filtered$clusternum = -1
train_filtered$wildtype = ''
for (i in unique(train_filtered$charcnt)) {
  train_filtered[train_filtered$charcnt == i, ]$clusternum <-
seq_cluster(aa(train_filtered[train_filtered$charcnt == i, ]$protein_sequence))
}

```

Step 4: Identify the wild type for train data !! long running query

for each protein sequence length(charcnt) and cluster number, loopthrough and find the protein sequence consensus (wildtype)

```{r}

for (i in unique(train_filtered$charcnt)) {
  for (j in unique (train_filtered[train_filtered$charcnt == i, ]$clusternum)){
    train_filtered[(train_filtered$charcnt == i & train_filtered$clusternum == j) , ]$wildtype <-
seq_consensus(aa(train_filtered[(train_filtered$charcnt == i & train_filtered$clusternum == j),
]$protein_sequence))
  }
}
```

Step 5: Order it by charcnt for ease of use

```{r}
train_filtered <- train_filtered[order(train_filtered$charcnt),]
```

Step 6: Add amino acid weightage for each sequence

```{r}

train_filtered_prop <- seq_stat_prop(aa(train_filtered$protein_sequence))
train_filtered_propdf <-  as.data.frame(do.call(rbind, train_filtered_prop))
train_filtered <- cbind (train_filtered,train_filtered_propdf )
```

Step 7: Add the group info to identify potential model training data

grouping train data by datasource, PH , charcnt and cluster note a set of charcnt and cluster belong to one wild type
```

```{r}
train_grouped <- train_filtered %>%     # Create ID by group
  group_by(charcnt,clusternum,pH,data_source) %>%
  dplyr::mutate(group = cur_group_id())


head(train_grouped)
```

Step 8: Final step to filter out only the top 25 groups to train our model

```{r}
train_grouped_top25 <- train_grouped %>%
  group_by(group) %>%
  filter(n() > 25)

head(train_grouped_top25)
```

Step 9: Add Mutation position data

```{r}

train_grouped_top25[,c('type','resid','WT','MUT')] = NA
for(i in 1:nrow(train_grouped_top25)){
if(train_grouped_top25$protein_sequence[i]==train_grouped_top25$wildtype[i]){
    train_grouped_top25[i ,c('type','resid','WT','MUT')] = as.list(c('WT',-1,NaN,NaN))
  # case 2 = substitution:
}
  else if(nchar(train_grouped_top25$protein_sequence[i])==nchar(train_grouped_top25$wildtype[i])){
    P <- mapply(function(x,y) which(x!=y)[1], strsplit(train_grouped_top25$protein_sequence[i],""),
strsplit(train_grouped_top25$wildtype[i],""))
    train_grouped_top25[i
,c('type','resid','WT','MUT')]=as.list(c('SUB',P,substr(train_grouped_top25$wildtype[i],P,P),substr(train_gro
uped_top25$protein_sequence[i],P,P)))
    # case 3 = deletion:
  } else if(nchar(train_grouped_top25$protein_sequence[i])<nchar(train_grouped_top25$wildtype[i])){
    wtsub <- substr(train_grouped_top25$wildtype[i],1,nchar(train_grouped_top25$protein_sequence[i]))
    P <- mapply(function(x,y) which(x!=y)[1], strsplit(train_grouped_top25$protein_sequence[i],""),
strsplit(wtsub,""))
   train_grouped_top25[i
,c('type','resid','WT','MUT')]=as.list(c('DEL',P,substr(train_grouped_top25$wildtype[i],P,P),NaN))
  }
}

```

Modifying Test Data

Step 1 Add amino acid weightage for each sequence: Let us create the sequence weightage for prediction

```{r}
test_prop <- seq_stat_prop(aa(test$protein_sequence))
test_propdf <-  as.data.frame(do.call(rbind, test_prop))
test <- cbind (test,test_propdf )

```

Step 2 Add mutation information to testing set:

Add mutation information to testing set:

```{r}
test[,c('type','resid','WT','MUT')] <- do.call(rbind,lapply(test$protein_sequence,function(seq){
  # case 1 = wild type:
  if(seq==wtseq){
    return(c('WT',-1,NaN,NaN))
  # case 2 = substitution:
  } else if(nchar(seq)==nchar(wtseq)){
    i <- mapply(function(x,y) which(x!=y)[1], strsplit(seq,""), strsplit(wtseq,""))
    return(c('SUB',i,substr(wtseq,i,i),substr(seq,i,i)))
  # case 3 = deletion:
  } else if(nchar(seq)<nchar(wtseq)){
    wtsub <- substr(wtseq,1,nchar(seq))
    i <- mapply(function(x,y) which(x!=y)[1], strsplit(seq,""), strsplit(wtsub,""))
    return(c('DEL',i,substr(wtseq,i,i),NaN))
  }
}))
head(test)
```

Step 3 - add the b factor from pdb file to test data

Read AlphaFold2 result for wild type sequence:

```{r}
pdb <- unique(test_pdb[test_pdb$V1=='ATOM',c(6,11)])
colnames(pdb) <- c('resid','b')
head(pdb)


# Add B factor to the testing set:
test_data_withbfactor <- merge(test,pdb,all.x=T)
test_data_withbfactor <- test_data_withbfactor[order(test_data_withbfactor$seq_id),]
head(test_data_withbfactor)


# Download blosum matrix and add score to testing set:
download.file('https://home.cc.umanitoba.ca/~psgendb/doc/local/pkg/ugene/data/weight_matrix/blosum100.
txt', destfile="BLOSUM100.txt")
blosum <- read.table('BLOSUM100.txt')
test_data_withbfactor$blosum <- apply(test_data_withbfactor,1,function(x){
  if(x['type']=='WT'){
```

```
    return(0)
  } else if(x['type']=='DEL'){
    return(-10)
  } else {
    return(blosum[x['WT'],x['MUT']])
  }
})
test_data_withbfactor$blosum[test_data_withbfactor$blosum>0] <- 0

head(test_data_withbfactor)
```

EDA

```{r}
hist(train$pH, xlab='pH Values',border = 'black',col = 'lightblue', main = 'Histogram of pH values in train dataset')
```

```{r}
train_gp = train_grouped_top25
head(train_gp)
```

Step 1 : Looking at the wild structure PDB Sequence shared in the data code

```{r}
protein_3d <- NGLVieweR("wildtype_structure_prediction_af2.pdb") %>%
  addRepresentation(
    type = "cartoon",
    param = list(
      backgroundColor = "black",
      colorScheme = "residueindex"
      )
       )  %>% setSpin()
 htmlwidgets::saveWidget(protein_3d, "protein_3d.html")
protein_3d

```

Step 2 : Making sure that the data is not repeated in both the Train and Test Datasets

```{r}
lapply(list(train, test), function(x) {
  paste(min(x[["seq_id"]]), max(x[["seq_id"]]))
})
```

Step 3: Checking the Summary statistics of the pH column to understand the patterns in the data

- The range of the pH should lie between [0:14]

```{r}
lapply(list(train, test), function(x){
  summary(x[["pH"]])
})
```

Step 4: Summary of column Data Source to understand the site which is most referred to understand the protein sequence

```{r}
lapply(list(train, test), function(x) {
  x %>%
    group_by(data_source) %>%
    summarise(
      .groups = "drop",
      n = n()
    ) %>%
    arrange(desc(n)) %>%
    head()
})

```

```{r}
eda_tm_1 <- train %>%
  group_by(tm) %>%
  summarise(
    .groups = "drop",
    n = n()
  ) %>%
  arrange(desc(n))

head(eda_tm_1,10)

```

Histograms for tm

```{r}
p01 <- ggplot(data = train, aes(x = tm)) +
  geom_histogram(bins = 100) +
  ylim(c(0,2000))

p02 <- ggplot(data = train, aes(x = tm)) +
  geom_histogram(bins = nrow(eda_tm_1)) +
  ylim(c(0,2000))

p01 + p02

```

Test Data : Insertions, Deletions and Substitutions

- All mutations in the test data are single point mutations. There are no insertions. Deletion *or* substitution occur only.

```{r}
res_exp_adist <- lapply(as.list(test[["protein_sequence"]]), function(x) {
  exp_adist <- adist(wtseq, x, counts = TRUE, partial = TRUE)

  dplyr::bind_cols(
    data.frame(exp_adist),
    data.frame(attributes(exp_adist)$counts),
    data.frame(attributes(exp_adist)$offsets)
  )
})
res_exp_adist <- dplyr::bind_rows(res_exp_adist)
names(res_exp_adist) <- gsub("X1.", "", names(res_exp_adist))

```

```{r}
res_exp_adist %>%
  group_by(ins, del, sub) %>%
  summarise(.groups = "drop", n=n())
```

```{r}
split_init <- unlist(strsplit(wtseq, ""))

```

```{r}
test_clean <- test %>%
  filter(
    protein_sequence != wtseq,
    stringi::stri_length(protein_sequence) == 221 # include substitutions only
    ) %>%
  rowwise() %>%
  mutate(
    sub_from = split_init[!(split_init == unlist(strsplit(protein_sequence, "")))],
    sub_to = unlist(strsplit(protein_sequence, ""))[!(split_init == unlist(strsplit(protein_sequence, "")))],
    sub_pos = which((split_init == unlist(strsplit(protein_sequence, ""))) == FALSE)
  ) %>%
  ungroup() %>%
  arrange(sub_from, sub_to, sub_pos) %>%
  mutate(
    tm = row_number()
  )

```

```{r}
```

```
head(test_clean,10)
```

```{r}
p1 <- ggplot(
    data = test_clean,
    mapping = aes(x = factor(sub_from, levels = sort(unique(sub_from))))
) +
geom_bar() +
ylim(c(0, 500)) +
xlab("sub_from") +
theme_minimal()

p2 <- ggplot(
    data = test_clean,
    mapping = aes(x = factor(sub_to, levels = sort(unique(sub_to))))
) +
geom_bar() +
ylim(c(0, 500)) +
xlab("sub_to") +
theme_minimal()

p_2 <- p1 + p2

ggsave(filename = "p_2.png", plot = p_2,  width = 10, dpi = 400)
```

```{r}
p3 <- lapply(
  split(1:221, ceiling(seq_along(1:221) / 25)), function(z) {
    if (length(z) < 25) {
      z <- min(z):(min(z) + 24)
    }
    ggplot(
      data = test_clean %>%
        filter(sub_pos %in% z),
      mapping = aes(x = factor(sub_pos, levels = as.character(z)))
    ) +
      geom_bar() +
      xlab("") +
      ylab("") +
      scale_x_discrete(drop = FALSE) +
      theme_minimal()
  }
)

p_3 <- wrap_plots(p3) +
  plot_layout(ncol = 2) +
  plot_annotation(
    title = "sub_pos"
  )
```

```
ggsave(filename = "p_3.png", plot = p_3,  width = 15, dpi = 400)
```

```{r}
head(train_gp)
```

```{r}
summary(train_gp$charcnt)
```

on Train Data

```{r}
p3t <- lapply(
  split(1:260, ceiling(seq_along(1:260) / 25)), function(z) {
    if (length(z) < 25) {
      z <- min(z):(min(z) + 24)
    }
    ggplot(
      data = train_gp %>%
        filter(resid %in% z),
      mapping = aes(x = factor(resid, levels = as.character(z)))
    ) +
      geom_bar() +
      xlab("") +
      ylab("") +
      scale_x_discrete(drop = FALSE) +
      theme_minimal()
  }
)

p_3t <- wrap_plots(p3t) +
  plot_layout(ncol = 2) +
  plot_annotation(
    title = "sub_pos"
  )

ggsave(filename = "p_3t.png", plot = p_3t,  width = 15, dpi = 600)


p3t2 <- lapply(
  split(276:450, ceiling(seq_along(276:450) / 25)), function(z) {
    if (length(z) < 25) {
      z <- min(z):(min(z) + 24)
    }
    ggplot(
      data = train_gp %>%
        filter(resid %in% z),
      mapping = aes(x = factor(resid, levels = as.character(z)))
    ) +
      geom_bar() +
```

```
      xlab("") +
      ylab("") +
      scale_x_discrete(drop = FALSE) +
      theme_minimal()
  }
)

p_3t2 <- wrap_plots(p3t2) +
  plot_layout(ncol = 2) +
  plot_annotation(
    title = "sub_pos"
  )
p_3t2
ggsave(filename = "p_3t2.png", plot = p_3t2,  width = 15, dpi = 600)
```

Create a Edge List

```{r}
edges_test_clean_1 <- test_clean %>%
  mutate(
    from = sub_from,
    to = sub_to
  ) %>%
  group_by(from, to) %>%
  summarise(
    .groups = "drop",
    edge_count = n()
  )


```

-sub_range_to_use is set to the first six positions at which a substitution occurs. Used as an example range for network exploration.

```{r}
sub_range_to_use = c(16:21)

edges_test_clean_2 <- test_clean %>%
  filter(sub_pos %in% sub_range_to_use) %>%
  mutate(
    from = paste0("wildtype_", sprintf("%03d", sub_pos), sub_from),
    to = paste0("mutation_", sprintf("%03d", sub_pos), sub_to)
  ) %>%
  group_by(from, to) %>%
  summarise(
    .groups = "drop",
    edge_count = n()
  )

```

Explore Nodes

The nodes occuring in wild type sequence and mutant sequences differ. No single point amino acid substitution changes from C, H or M to another amino acid. In contrast a amino acid substitution can go to C, H or M.

```{r}
sort(unique(edges_test_clean_1$from))
sort(unique(edges_test_clean_1$to))
```

- Amino acids that occure in the wild type sequence but not in the mutant sequences.

```{r}
setdiff(
  sort(unique(edges_test_clean_1$from)),
  sort(unique(edges_test_clean_1$to))
)
```

- Amino acids that occure in the mutant sequences but not in the wild type sequence.

```{r}
setdiff(
  sort(unique(edges_test_clean_1$to)),
  sort(unique(edges_test_clean_1$from))
)
```

Set Levels

Get all unique amino acids of wild type and mutant sequences.

```{r}
edge_levels <- sort(unique(c(edges_test_clean_1$from, edges_test_clean_1$to)))
```

Visualize Edges

There are no edges from C, H or M

```{r}
p4 <- ggplot(
  data = edges_test_clean_1 %>%
    mutate(
      from = factor(from, levels = edge_levels),
      to = factor(to, levels = edge_levels)
    ),
  aes(x = to, y = from)
) +
  geom_point(aes(size = edge_count)) +
```

```
  scale_x_discrete(drop = FALSE) +
  scale_y_discrete(drop = FALSE) +
  theme_minimal()

p_4 <- p4 # just for coherence with other plots

p_4
# ggsave(filename = "p_4.png", plot = p_4,  width = 10, dpi = 400)
```

Node graph for grouped Train data

```{r}
edges_clean_train_1 <- train_grouped_top25 %>%
  mutate(
    from = WT,
    to = MUT
  ) %>%
  group_by(from, to) %>%
  summarise(
    .groups = "drop",
    edge_count = n()
  )
```

```{r}
sort(unique(edges_clean_train_1$from))
sort(unique(edges_clean_train_1$to))
```

```{r}
setdiff(
  sort(unique(edges_clean_train_1$from)),
  sort(unique(edges_clean_train_1$to))
)
```

```{r}
setdiff(
  sort(unique(edges_clean_train_1$to)),
  sort(unique(edges_clean_train_1$from))
)
```

```{r}
edge_levels1 <- sort(unique(c(edges_clean_train_1$from, edges_clean_train_1$to)))
```

```{r}
p6 <- ggplot(
```

```
  data = edges_clean_train_1 %>%
    mutate(
      from = factor(from, levels = edge_levels1),
      to = factor(to, levels = edge_levels1)
    ),
  aes(x = to, y = from)
) +
  geom_point(aes(size = edge_count)) +
  scale_x_discrete(drop = FALSE) +
  scale_y_discrete(drop = FALSE) +
  theme_minimal()

p_6 <- p6 # just for coherence with other plots

p_6
#ggsave(filename = "p_6.png", plot = p_6,  width = 10, dpi = 400)
```

Functions

```{r}
protein_sequence_to_edge_list <- function(ps, sub_range = NULL) {
  if(is.null(sub_range)) print("No sub_range provided. Full sequence is used.")
  if(is.null(sub_range)) {
      inner <- unlist(strsplit(ps, ""))
  } else {
      inner <- unlist(strsplit(ps, ""))[sub_range]
  }
  inner <- paste0("wildtype_", sprintf("%03d", sub_range), inner)
  # create artificial root
  e0 <- data.frame(
    "from" = "zero_0000",
    "to" = inner,
    "edge_count" = 1L
  )
  # create edges
  e1 <- data.frame(
    "from" = inner[1:length(inner) -1],
    "to" = inner[2:length(inner)],
    "edge_count" = 1L
  )
  # use for cyclic graph: connect start to end
  # e2 <- data.frame(
  #   "from" = inner[length(inner)],
  #   "to" = inner[1],
  #   "edge_count" = 1L
  # )
  bind_rows(
    e0,
    e1#,
    # e2
    )
```

```
}
```

Visualize Networks

```{r}
edges_seq_init <- protein_sequence_to_edge_list(ps = wtseq, sub_range = sub_range_to_use)
```

-Get positions of the wild type sequence without any mutation (i.e., substitution)

```{r}
seq_init_to_none <- edges_seq_init$to[!(edges_seq_init$to %in% edges_test_clean_2$from)]
```

Create artificial edges for non-mutated positions.

```{r}
edges_seq_init_to_none <- data.frame(
  "from" = ifelse(length(seq_init_to_none) == 0, "zero_0000", seq_init_to_none),
  "to" = "none",
  "edge_count" = 1L
)
```

Combine all edges. Optional: Clean for further exploration.

```{r}
edges_combined <- bind_rows(
  edges_test_clean_2,
  edges_seq_init,
  edges_seq_init_to_none
) %>%
  filter(
    !(stringr::str_detect(from, "wildtype") & stringr::str_detect(to, "wildtype"))
  )
```

```{r}
head(edges_combined)
glimpse(edges_combined)
```

Turn edge list into a graph opbject. Optional: modify and cleanup graph object

```{r}
g1 <- as_tbl_graph(edges_combined)
g1 <- g1 %>%
  activate(nodes) %>%
  mutate(label = stringr::str_sub(name, -4, -1)) %>%
  filter(name != "none")
g1
```

```
```

Create network plot

```{r}
p_g1 <- ggraph(g1, layout = "dendrogram", circular = TRUE) +
  geom_edge_diagonal() +
  geom_node_point() +
  # TODO: adjust angle for labels
  geom_node_label(mapping = aes(label = label), size = 4, alpha = .9, vjust = .5) #, repel = TRUE)
```

```{r}
ggsave(filename = "p_g1.png", plot = p_g1,  width = 10, height = 10, dpi = 400)
p_g1
```

Modelling Predicting Tm

XGBoost

```{r}
set.seed(200)
library(xgboost)
trainset <- sample(1:nrow(train_grouped_top25), 1100)  # DO NOT CHANGE: you must sample 80 data
points for training
validset <- setdiff(1:nrow(train_grouped_top25), trainset)  # The remaining is used for validation
#filter the necessary rows for modeling from train dataframe
traingrp <- train_grouped_top25 %>%
dplyr::select(pH,tm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
traingrp$resid = as.integer(traingrp$resid)
summary(traingrp)
# group var won't filter so I'm talking it out here
traingrp = traingrp[c(-1)]
# makign sure it's a dataframe for the matrix
traingrp = as.data.frame(traingrp)
head(traingrp)

#filter test rows similar to train data
test_data_withbfactor$tm <- 0
testxgb <- test_data_withbfactor %>%
dplyr::select(pH,tm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

testxgb = as.data.frame(testxgb)
str(testxgb)
xgb_train = xgb.DMatrix(data=(data.matrix(traingrp[trainset,-2])), label=(traingrp[trainset,2] ))
xgb_valid = xgb.DMatrix(data=(data.matrix(traingrp[validset,-2])), label=(traingrp[validset,2] ))
xgb_test = xgb.DMatrix(data=(data.matrix(testxgb[,-2])), label=(testxgb[,2] ))
xgb <- xgboost(data = xgb_train, max.depth=5,nrounds=25)
importance_matrix = xgb.importance(colnames(xgb_train), model = xgb)
importance_matrix
```

```
pred_xgb = predict(xgb, xgb_test)
head(pred_xgb)
df = data.frame( seq_id  = test[, "seq_id"],
                 type = test[,"type"],
                 Tm = pred_xgb )
submission_TM <-  data.frame(seq_id = test$seq_id)
submission_TM$tm <- (-
rank(test_data_withbfactor$b)/length(submission_TM$seq_id))+(rank(test_data_withbfactor$blosum)/lengt
h(submission_TM$seq_id)+(rank(pred_xgb))/length(submission_TM$seq_id))

pred_xgb_valid = predict(xgb, xgb_valid)
train_mse_OLS_XGB = mean((pred_xgb_valid - traingrp[validset,"tm"])^2)
train_mse_OLS_XGB


# write.csv(df,"type_included.csv")
 #write.csv(submission_TM ,"submissions_TM.csv")
```

```
### Testing All Other Models on Tm

```{r}


set.seed(200)
traingrp2 = na.omit(traingrp)
trainset <- sample(1:nrow(traingrp2), 1100)
validset <- setdiff(1:nrow(traingrp2), trainset)


library(randomForest)


 rfo <- randomForest(tm ~., data = traingrp2[trainset,])
pred_rfo <- predict(rfo, newdata = traingrp2[validset,], type='response')


train_mse_OLS_rf = mean((pred_rfo- traingrp2[validset,"tm"])^2)

train_mse_OLS_rf


rp = rpart(tm ~., data = traingrp2[trainset,])

pred_rp = predict(rp, newdata = traingrp2[validset,])



train_mse_OLS_rp = mean((pred_rp- traingrp2[validset,"tm"])^2)

train_mse_OLS_rp
```
```

```
trainggrp2CT = traingrp2


trainggrp2CT$WT = as.factor(trainggrp2CT$WT )
trainggrp2CT$MUT = as.factor(trainggrp2CT$MUT )
Party_mod = ctree(tm ~., data = trainggrp2CT[trainset,])

ctree_pred = predict(Party_mod, trainggrp2CT[validset,])

train_mse_OLS_ctree= mean((ctree_pred- trainggrp2CT[validset,"tm"])^2)

train_mse_OLS_ctree



# need to convert it like this back from factor for it to work
Tree = tree(tm ~., data = traingrp2[trainset,])

Tree_pred = predict(Tree, traingrp2[validset,])

train_mse_OLS_tree= mean((Tree_pred- traingrp2[validset,"tm"])^2)

train_mse_OLS_tree

## GBM


gbm = gbm.fit.final3 <- gbm(
  formula = tm ~.,
  distribution = "gaussian",
  n.trees = 1000,
  data = trainggrp2CT[trainset,]
  )

gbm_pred = predict(gbm, trainggrp2CT[validset,])

train_mse_OLS_gbm = mean((gbm_pred- trainggrp2CT[validset,"tm"])^2)

train_mse_OLS_gbm

summary.gbm(gbm)

```

modelling on dtm

Creating dtm on our groups

```{r}

### Taking means of groups with same wildtype then trying to identify the change that the mutation makes
```

```
train_grouped_top25_dtm = train_grouped_top25
train_grouped_top25_dtm$dtm=0
for (i in unique(train_grouped_top25_dtm$group)) {
  grp_mean_tm = mean(train_grouped_top25_dtm[train_grouped_top25_dtm$group == i,]$tm)
  train_grouped_top25_dtm[train_grouped_top25_dtm$group == i,]$dtm <-
(train_grouped_top25_dtm[train_grouped_top25_dtm$group == i,]$tm - grp_mean_tm)
}

train_grouped_top25_dtm$protein_length = str_length(train_grouped_top25_dtm$protein_sequence)
train_grouped_top25_dtm$WT_length = str_length(train_grouped_top25_dtm$wildtype)

train_grouped_top25_dtm_normalized = train_grouped_top25_dtm
train_grouped_top25_dtm_normalized$dtm <- ave(train_grouped_top25_dtm_normalized$dtm,
train_grouped_top25_dtm_normalized$group, FUN=function(x) scale(x))
## Putting string length on it too
```

Pulling in Jin Data to help

```{r}

## Data From Jin in Kaggle: https://github.com/JinyuanSun/mutation-stability-data

JinTrain = read.csv("train_jin.csv")

JinTrain$resid = JinTrain$position

JinTest = read.csv("test_jin.csv")

JinTest$resid = JinTest$position

JinTm = read.csv("tm_jin.csv")

JinTm$dtm = JinTm$dTm

JinTm$resid = JinTm$position

# bio seqing on all the Jin data
JinTm_prop <- seq_stat_prop(aa(JinTm$mutant_seq))
JinTm_propdf <-  as.data.frame(do.call(rbind, JinTm_prop))
JinTm <- cbind (JinTm,JinTm_propdf )

JinTrain_prop <- seq_stat_prop(aa(JinTrain$mutant_seq))
JinTrain_propdf <-  as.data.frame(do.call(rbind, JinTrain_prop))
JinTrain <- cbind (JinTrain,JinTrain_propdf )


JinTest_prop <- seq_stat_prop(aa(JinTest$mutant_seq))
JinTest_propdf <-  as.data.frame(do.call(rbind, JinTest_prop))
JinTest <- cbind (JinTest,JinTest_propdf )

```

Putting Jin data together

```{r}

JinTest$protein_length = str_length(JinTest$mutant_seq)
JinTest$WT_length = str_length(JinTest$sequence)
#normalizing dGG for each group
JinTest$dtm <- ave(JinTest$ddG, JinTest$PDB, FUN=function(x) scale(x))
#df
#JinTest$dTm = scale(JinTest$ddG)

#Renaming Columns so they can be joined
names(JinTest)[names(JinTest) == "mutation"] <- "MUT"
names(JinTest)[names(JinTest) == "wildtype"] <- "WT"

# Getting rid of mutations with only one row

JinTrain$protein_length = str_length(JinTrain$mutant_seq)
JinTrain$WT_length = str_length(JinTrain$sequence)
#normalizing dGG
JinTrain$dtm <- ave(JinTrain$ddG, JinTrain$PDB, FUN=function(x) scale(x))
#Renaming Columns so they can be joined
names(JinTrain)[names(JinTrain) == "mutation"] <- "MUT"
names(JinTrain)[names(JinTrain) == "wildtype"] <- "WT"

JinTm$protein_length = str_length(JinTm$mutant_seq)
JinTm$WT_length = str_length(JinTm$sequence)
JinTm_scaled = JinTm
JinTm_scaled$dtm <- ave(JinTm$dTm, JinTm$PDB, FUN=function(x) scale(x))

# Getting rid of mutations with only one row
JinTm = na.omit(JinTm)
JinTm_scaled = na.omit(JinTm_scaled)

## combined all Jin data
JinTm_scaled =  JinTm_scaled %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT, protein_length,
WT_length)
JinTrain2 =  JinTrain %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT, protein_length,
WT_length)
JinTest2  = JinTest %>% dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT,
protein_length, WT_length)
Super_jin = rbind(JinTest2, JinTrain2, JinTm_scaled)
Super_jin = na.omit(Super_jin)
```

Putting together dTm and Normalized dTm files

```{r}
```

```
traingrp_dtm <- train_grouped_top25_dtm %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
traingrp_dtm = traingrp_dtm[c(-1)]

JindTm <- JinTm %>% dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

Super_jin = Super_jin %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
dTm_mastertable = rbind(traingrp_dtm, JindTm)


train_grouped_top25_dtm_normalized  = train_grouped_top25_dtm_normalized %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
train_grouped_top25_dtm_normalized = train_grouped_top25_dtm_normalized[c(-1)]
Ultra_dataset = rbind(train_grouped_top25_dtm_normalized, Super_jin)
#dtm_master_file = rbind(JinTm, train_grouped_top25_dtm )
```

Modelling Stuff dTm

```{r}
set.seed(200)
library(xgboost)
trainset <- sample(1:nrow(dTm_mastertable), 1700)  # DO NOT CHANGE: you must sample 80 data points for training
validset <- setdiff(1:nrow(dTm_mastertable), trainset)  # The remaining is used for validation
#filter the necessary rows for modeling from train dataframe
dTm_mastertable$resid = as.integer(dTm_mastertable$resid)

# makign sure it's a dataframe for the matrix
dTm_mastertable = as.data.frame(dTm_mastertable)
head(dTm_mastertable)

#filter test rows similar to train data
test_data_withbfactor$dtm <- 0
test_data_withbfactor$protein_length = str_length(test_data_withbfactor$protein_sequence)
test_data_withbfactor$WT_length = str_length(wtseq)
testxgb <- test_data_withbfactor %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

testxgb = as.data.frame(testxgb)
str(testxgb)
xgb_train = xgb.DMatrix(data=(data.matrix(dTm_mastertable[trainset,-1])),
label=(dTm_mastertable[trainset,1] ))
xgb_valid = xgb.DMatrix(data=(data.matrix(dTm_mastertable[validset,-1])),
label=(dTm_mastertable[validset,1] ))
xgb_test = xgb.DMatrix(data=(data.matrix(testxgb[,-1])), label=(testxgb[,1] ))
xgb <- xgboost(data = xgb_train, nrounds = 25)
importance_matrix = xgb.importance(colnames(xgb_train), model = xgb)
importance_matrix
pred_xgb = predict(xgb, xgb_test)
```

```
head(pred_xgb)
df_XGB = data.frame( seq_id  = test[, "seq_id"],
                type = test[,"type"],
                dTm = pred_xgb )
submission_dTM <-  data.frame(seq_id = test$seq_id)
submission_dTM$tm <- (-
rank(test_data_withbfactor$b)/length(submission_dTM$seq_id))+(rank(test_data_withbfactor$blosum)/leng
th(submission_dTM$seq_id)+(rank(pred_xgb))/length(submission_dTM$seq_id))


pred_xgb_valid = predict(xgb, xgb_valid)
train_mse_OLS_XGB_dtm  = mean((pred_xgb_valid - dTm_mastertable[validset,"dtm"])^2)
train_mse_OLS_XGB_dtm
                 #+(rank(pred_xgb))/length(submission$seq_id))

 # write.csv(df_XGB_dtm ,"type_included.csv")
 write.csv(submission_dTM,"submissions_dTm.csv")
```

```
## Different Model for dTm
```

```{r}


set.seed(200)

dTm_mastertable2 = na.omit(dTm_mastertable)
dTm_mastertable2$resid = as.integer(dTm_mastertable2$resid)
trainset <- sample(1:nrow(dTm_mastertable2), 1100)  # DO NOT CHANGE: you must sample 80 data
points for training
validset <- setdiff(1:nrow(dTm_mastertable2), trainset)  # The remaining is used for validation

library(randomForest)

 rfo <- randomForest(dtm ~., data = dTm_mastertable2[trainset,])
pred_rfo <- predict(rfo, newdata = dTm_mastertable2[validset,], type='response')


train_mse_OLS_rf_dtm  = mean((pred_rfo- dTm_mastertable2[validset,"dtm"])^2)

train_mse_OLS_rf_dtm


rp = rpart(dtm ~., data = dTm_mastertable2[trainset,])

pred_rp  = predict(rp, newdata = dTm_mastertable2[validset,])



train_mse_OLS_rp_dtm  = mean((pred_rp- dTm_mastertable2[validset,"dtm"])^2)
```

```
train_mse_OLS_rp_dtm

trainggrp2CT = dTm_mastertable2
# need to convert it like this back from factor for it to work
trainggrp2CT$WT = as.factor(trainggrp2CT$WT )
trainggrp2CT$MUT = as.factor(trainggrp2CT$MUT )
Party_mod = ctree(dtm ~., data = trainggrp2CT[trainset,])

ctree_pred = predict(Party_mod, trainggrp2CT[validset,])

train_mse_OLS_ctree_dtm = mean((ctree_pred- trainggrp2CT[validset,"dtm"])^2)

train_mse_OLS_ctree_dtm



# need to convert it like this back from factor for it to work
Tree = tree(dtm ~., data = dTm_mastertable2[trainset,])

Tree_pred = predict(Tree, dTm_mastertable2[validset,])

train_mse_OLS_tree_dtm  = mean((Tree_pred- dTm_mastertable2[validset,"dtm"])^2)

train_mse_OLS_tree_dtm

## GBM


gbm = gbm.fit.final3 <- gbm(
  formula = dtm ~.,
  distribution = "gaussian",
  n.trees = 1000,
  data = trainggrp2CT[trainset,]
  )

gbm_pred = predict(gbm, trainggrp2CT[validset,])

train_mse_OLS_gbm_dtm  = mean((gbm_pred- trainggrp2CT[validset,"dtm"])^2)

train_mse_OLS_gbm_dtm

summary.gbm(gbm)

```

Modelling Stuff Normalized dTm

```{r}
set.seed(200)
library(xgboost)
Ultra_dataset = na.omit(Ultra_dataset)
trainset <- sample(1:nrow(Ultra_dataset), 4700)
```

```
validset <- setdiff(1:nrow(Ultra_dataset), trainset)
#filter the necessary rows for modeling from train dataframe
Ultra_dataset$resid = as.integer(Ultra_dataset$resid)

# makign sure it's a dataframe for the matrix
Ultra_dataset = as.data.frame(Ultra_dataset)
Ultra_dataset$dtm = round(Ultra_dataset$dtm,10)
summary(Ultra_dataset)

#filter test rows similar to train data
test_data_withbfactor$dtm <- 0
test_data_withbfactor$resid = as.integer(test_data_withbfactor$resid)
test_data_withbfactor$protein_length = str_length(test_data_withbfactor$protein_sequence)
test_data_withbfactor$WT_length = str_length(wtseq)
testxgb <- test_data_withbfactor %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

testxgb = as.data.frame(testxgb)
str(testxgb)
xgb_train = xgb.DMatrix(data=(data.matrix(Ultra_dataset[trainset,-1])), label=(Ultra_dataset[trainset,1] ))
xgb_valid = xgb.DMatrix(data=(data.matrix(Ultra_dataset[validset,-1])), label=(Ultra_dataset[validset,1] ))
xgb_test = xgb.DMatrix(data=(data.matrix(testxgb[,-1])), label=(testxgb[,1] ))

xgb <- xgboost(data = xgb_train, max.depth=5,nrounds=1000)

cv <- xgb.cv(data = xgb_train, nrounds = 50, nthread = 2, nfold = 25, metrics = list("rmse"),
             max_depth = 3, eta = 1, objective = "reg:squarederror")

importance_matrix = xgb.importance(colnames(xgb_train), model = xgb)
importance_matrix

pred_xgb_valid_dtm = predict(xgb, xgb_valid)

train_mse_OLS_XGB_dtm_nor  = mean((pred_xgb_valid_dtm - Ultra_dataset[validset,"dtm"])^2)
train_mse_OLS_XGB_dtm_nor



## Different Model for Normalized dTm
```

All Other Different Models

```{r}


set.seed(200)  #
# DO NOT MODIFY the next four lines
# 1 means good quality, 0 means bad quality
Ultra_dataset2 = na.omit(Ultra_dataset)
trainset <- sample(1:nrow(Ultra_dataset2), 4700)  # DO NOT CHANGE: you must sample 80 data points
for training
```

```
validset <- setdiff(1:nrow(Ultra_dataset2), trainset)  # The remaining is used for validation

library(randomForest)

rfo <- randomForest(dtm ~., data = Ultra_dataset2[trainset,])
pred_rfo <- predict(rfo, newdata = Ultra_dataset2[validset,], type='response')

pred_rfo_test <- predict(rfo, newdata = testxgb, type='response')
df_rf_nor= data.frame( seq_id  = test[, "seq_id"],
                type = test[,"type"],
                dTm = pred_rfo_test )



submission_rf <-  data.frame(seq_id = test$seq_id)
submission_rf$tm <- (-
rank(test_data_withbfactor$b)/length(submission_rf$seq_id))+(rank(test_data_withbfactor$blosum)/length(
submission_rf$seq_id)+(rank(pred_rfo_test))/length(submission_rf$seq_id))

train_mse_OLS_rf_dtm_nor   = mean((pred_rfo- Ultra_dataset2[validset,"dtm"])^2)

train_mse_OLS_rf_dtm_nor


rp = rpart(dtm ~., data = Ultra_dataset2[trainset,])

pred_rp  = predict(rp, newdata = Ultra_dataset2[validset,])


train_mse_OLS_rp_dtm_nor   = mean((pred_rp- Ultra_dataset2[validset,"dtm"])^2)

train_mse_OLS_rp_dtm_nor

trainggrp2CT = Ultra_dataset2

trainggrp2CT$WT = as.factor(trainggrp2CT$WT )
trainggrp2CT$MUT = as.factor(trainggrp2CT$MUT )
Party_mod = ctree(dtm ~., data = trainggrp2CT[trainset,])

ctree_pred = predict(Party_mod, trainggrp2CT[validset,])



train_mse_OLS_ctree_dtm_nor  = mean((ctree_pred- trainggrp2CT[validset,"dtm"])^2)

train_mse_OLS_ctree_dtm_nor


Tree = tree(dtm ~., data = Ultra_dataset2[trainset,])

Tree_pred = predict(Tree, Ultra_dataset2[validset,])
```

```
Tree_pred_test = predict(Tree, newdata = testxgb)

train_mse_OLS_tree_dtm_nor   = mean((Tree_pred- Ultra_dataset2[validset,"dtm"])^2)

train_mse_OLS_tree_dtm_nor

## GBM


gbm = gbm.fit.final3 <- gbm(
  formula = dtm ~.,
  distribution = "gaussian",
  n.trees = 1000,
  data = trainggrp2CT[trainset,]
  )

gbm_pred = predict(gbm, trainggrp2CT[validset,])

train_mse_OLS_gbm_dtm_nor   = mean((gbm_pred- trainggrp2CT[validset,"dtm"])^2)

train_mse_OLS_gbm_dtm_nor

summary.gbm(gbm)



```

Final Models

Tm Trained with most data possible : XGB Chosen. Using full data to train.

```
```{r}
set.seed(200)
#filter the necessary rows for modeling from train dataframe
traingrp <- train_grouped_top25 %>%
dplyr::select(pH,tm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
traingrp$resid = as.integer(traingrp$resid)
# group var won't filter so I'm talking it out here
traingrp = traingrp[c(-1)]
# makign sure it's a dataframe for the matrix
traingrp = as.data.frame(traingrp)

#filter test rows similar to train data
test_data_withbfactor$tm <- 0
testxgb <- test_data_withbfactor %>%
dplyr::select(pH,tm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

testxgb = as.data.frame(testxgb)
#Setting up train for XGB training
xgb_train = xgb.DMatrix(data=(data.matrix(traingrp[,-2])), label=(traingrp[,2] ))
xgb_test = xgb.DMatrix(data=(data.matrix(testxgb[,-2])), label=(testxgb[,2] ))
```
```

```
# Using Cross Validation for best parameter
xgbcv = xgb.cv(data = xgb_train, nfold =25, nrounds =1000, early_stopping_rounds = 40)

opt_iterations = xgbcv$best_iteration

xgb <- xgboost(data = xgb_train, max.depth=5,nrounds=opt_iterations)

pred_xgb = predict(xgb, xgb_test)
head(pred_xgb)
df = data.frame( seq_id  = test[, "seq_id"],
                 type = test[,"type"],
                 Tm = pred_xgb )
submission_TM <-  data.frame(seq_id = test$seq_id)
submission_TM$tm <- (-
rank(test_data_withbfactor$b)/length(submission_TM$seq_id))+(rank(test_data_withbfactor$blosum)/lengt
h(submission_TM$seq_id)+(rank(pred_xgb))/length(submission_TM$seq_id))

importance_matrix = xgb.importance(colnames(xgb_train), model = xgb)
importance_matrix

print(xgb.plot.importance(importance_matrix = importance_matrix[1:10]))
                 #+(rank(pred_xgb))/length(submission$seq_id))
df_submission = data.frame( seq_id  = test[, "seq_id"],
                 type = test[,"type"],
                 Tm = submission_TM$tm )

# for deletions we will only use non-modelling data
# Deletion type:
idx <- df_submission$type  == 'DEL'
df_submission_del = df_submission
df_submission_del[idx,'Tm'] <-(-
rank(test_data_withbfactor$b[idx])/length(submission_TM$seq_id[idx]))+(rank(test_data_withbfactor$blos
um[idx])/length(submission_TM$seq_id[idx]) )



# write.csv(df,"type_included.csv")
 write.csv(df_submission_del ,"submissions_TM_DEL.csv")
  write.csv(df_submission ,"submissions_TM.csv")
```

dTm Model Chosen: XGB

```{r}
set.seed(200)
library(xgboost)


#filter the necessary rows for modeling from train dataframe
dTm_mastertable$resid = as.integer(dTm_mastertable$resid)
```

```
# makign sure it's a dataframe for the matrix
 dTm_mastertable = dTm_mastertable %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)
 dTm_mastertable = as.data.frame(dTm_mastertable)
dTm_mastertable = na.omit(dTm_mastertable)
#filter test rows similar to train data
test_data_withbfactor$dtm <- 0
testxgb <- test_data_withbfactor %>%
dplyr::select(dtm,A,C,G,T,W,S,M,K,R,Y,B,D,H,V,N,E,F,I,L,P,Q,resid,WT,MUT)

testxgb = as.data.frame(testxgb)

xgb_train = xgb.DMatrix(data=(data.matrix(dTm_mastertable[,-1])), label=(dTm_mastertable[,1] ))
xgb_test = xgb.DMatrix(data=(data.matrix(testxgb[,-1])), label=(testxgb[,1] ))

xgbcv_dtm = xgb.cv(data = xgb_train, nfold =25, nrounds =1000, early_stopping_rounds = 40)

xgbcv_dtm$best_iteration

opt_iterations_dtm = xgbcv_dtm$best_iteration

xgb_dtm  <- xgboost(data = xgb_train, max.depth=5,nrounds=opt_iterations_dtm)

importance_matrix = xgb.importance(colnames(xgb_train), model = xgb_dtm )

importance_matrix

print(xgb.plot.importance(importance_matrix = importance_matrix[1:10]))

pred_xgb_dtm  = predict(xgb_dtm , xgb_test)

df_XGB_dtm  = data.frame( seq_id  = test[, "seq_id"],
              type = test[,"type"],
              dTm = pred_xgb_dtm  )
submission_dTM <-  data.frame(seq_id = test$seq_id)
submission_dTM$tm <- (-
rank(test_data_withbfactor$b)/length(submission_dTM$seq_id))+(rank(test_data_withbfactor$blosum)/leng
th(submission_dTM$seq_id)+(rank(pred_xgb_dtm ))/length(submission_dTM$seq_id))

               #+(rank(pred_xgb))/length(submission$seq_id))

idx <- submission_dTM$type  == 'DEL'
df_submission_del_dTM = submission_dTM
df_submission_del_dTM[idx,'Tm'] <-(-
rank(test_data_withbfactor$b[idx])/length(submission_TM$seq_id[idx]))+(rank(test_data_withbfactor$blos
um[idx])/length(submission_TM$seq_id[idx]) )

  write.csv(df_submission_del_dTM ,"subsmissions_dTm.csv")
  write.csv(submission_dTM,"submissions_dTm.csv")

```

dTm Normalized (I guess it's just target at this point)

```{r}

mtry <- tuneRF(Ultra_dataset2[-1],Ultra_dataset2$dtm, ntreeTry=500,
               stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
print(mtry)
print(best.m)

summary(Ultra_dataset2)
rfo_dtm <- randomForest(dtm ~., data = Ultra_dataset2, mtry = best.m)


pred_rfo_test_dtm <- predict(rfo_dtm, newdata = testxgb, type='response')
df_rf_nor= data.frame( seq_id  = test[, "seq_id"],
                type = test[,"type"],
                dTm = pred_rfo_test_dtm )
write.csv(df_rf_nor ,"type_included_nor_rf.csv")


submission_rf_dtm<-  data.frame(seq_id = test$seq_id)
submission_rf_dtm$tm <- (-
rank(test_data_withbfactor$b)/length(submission_rf_dtm$seq_id))+(rank(test_data_withbfactor$blosum)/le
ngth(submission_rf_dtm$seq_id)+(rank(pred_rfo_test))/length(submission_rf_dtm$seq_id))

idx <- submission_rf_dtm$type  == 'DEL'
df_submission_del_dTM_nor = submission_rf_dtm
df_submission_del_dTM_nor[idx,'tm'] <-(-
rank(test_data_withbfactor$b[idx])/length(df_submission_del_dTM_nor$seq_id[idx]))+(rank(test_data_wit
hbfactor$blosum[idx])/length(df_submission_del_dTM_nor$seq_id[idx]) )

write.csv(df_submission_del_dTM_nor,"RF_combo_dtm_nor.csv")
```

5. References

1. https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6128648/
3. https://en.wikipedia.org/wiki/BLOSUM
4. http://thegrantlab.org/bio3d/articles/online/pdb_vignette/Bio3D_pdb.html
5. https://proteinstructures.com/structure/protein-databank/
6. https://github.com/JinyuanSun/mutation-stability-data
7. https://www.kaggle.com/code/pjt222/sequence-as-graph

8. https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction/discussion/371747
9. http://thegrantlab.org/bio3d/articles/online/pdb_vignette/Bio3D_pdb.html
10. https://cran.r-project.org/web/packages/protr/vignettes/protr.html#6_similarity_calculation_by_sequence_alignment
11. https://cran.r-project.org/web/packages/bioseq/index.html