HIDS 7006: Final Project Report

Prediction of SARS-COVID19 severity based on graph-based features of T-cell receptors’ CDR3 amino acid sequences

by Aizhan Uteubayeva
(NetID: au198)

TABLE OF CONTENT

1. INTRODUCTION

1.1. Translating Biological Data into Clinical Outcomes

The challenge of translating biological data into actionable clinical outcomes is significant but holds immense potential. Precision medicine aims to delve into individual patient data to gain insights into the roots of health problems, enabling personalized treatment plans. In the context of COVID-19, understanding the severity of the disease is crucial as it is closely linked to patient hospitalizations and mortality. The severity of COVID-19 often determines the course of treatment, the level of care required, and the likelihood of severe outcomes, including hospitalization and death.

1.2. Why TCR Sequences?

T cell receptors (TCRs) play a critical role in the immune response by recognizing antigens presented by infected cells. The diversity of TCRs, especially in the complementarity determining region 3 (CDR3) loop, is vital for antigen binding. This diversity is generated through the rearrangement of V (variable), D (diversity), and J (joining) region sequence elements during lymphocyte maturation, resulting in extensive length and sequence heterogeneity. CDR3 loops provide the principal peptide-binding residues in TCRs, making them essential for the immune system's ability to recognize a wide variety of antigens (NCBI, 2022a).

The TCR is a complex protein on the surface of T cells, which are a type of white blood cell involved in the adaptive immune response. Each TCR consists of two different polypeptide chains: alpha (TRA) and beta (TRB). These chains are encoded by separate genes that undergo V(D)J recombination during T cell development, a process that generates a vast repertoire of unique TCRs, each with distinct antigen-binding specificities (Frontiers in Physics, 2022).

In TCR sequences, the alpha (TRA) and beta (TRB) chains are particularly important. The alpha chain provides part of the specificity for the antigen, while the beta chain works alongside the alpha chain to recognize antigens presented by major histocompatibility complex (MHC) molecules. The variability in the CDR3 region of these chains is crucial for the TCR's specificity and its ability to mount an effective immune response (Cornell, 2022).

The CDR3 region is the most variable part of the TCR and directly contacts the peptide-MHC complex. This region's diversity is critical for the immune system's capacity to recognize a broad array of antigens. The TRA and TRB chains' CDR3 regions contribute differently to antigen recognition: the alpha chain typically provides a more stable structural framework, while the beta chain offers a broader diversity in antigen recognition (ArXiv, 2018).

In this project, advanced computational tools such as the AAPLZGraph are employed to analyze the structural features of TCR sequences. By extracting node-level features from graph networks, the aim is to capture the intricate patterns within the TCR repertoire that are essential for effective immune responses (Taneja, 2022). Additionally, leveraging clinical data, genetic information, and sequence alignment based on COVID-19 peptide structures provides a comprehensive approach to understanding disease mechanisms and predicting disease severity (NCBI, 2022b).

1.3. Computational Approach: LZGraph

Analyzing high-dimensional TCR sequence data requires substantial computational power. The LZGraph package is employed to create graphs where nodes represent unique sequences (clonotypes) and edges represent similarities or functional relationships between these sequences. Centralities like eigenvector centrality are used to measure the influence of nodes within the network, providing insights into the structural and relational dynamics of TCRB sequences. This network analysis helps identify key sequences and understand the complexity and diversity of the immune response in severe COVID-19 cases.

1.4. Objective and Dataset Selection

Initially, the project aimed to analyze TCR sequences to predict disease severity based on a dataset from a research paper titles as "Severely ill COVID-19 patients display impaired exhaustion features in SARS-CoV-2-reactive CD8+ T cells" (PMID: 33478949, PMCID: PMC8101257, DOI: 10.1126/sciimmunol.abe4782) by Kusnadi et al., 2021. However, due to the small sample size, another dataset was chosen from the paper - “Machine learning identifies T cell receptor repertoire signatures associated with COVID-19 severity”' (PMID: 36670287, PMCID: PMC9853487, DOI: 10.1038/s42003-023-04447-4) by Park et al., 2023, which includes 270 patients with clinical outcomes and severity data. Despite the change in datasets, the objective remained to engineer a graph-based approach for feature extraction to predict the severity of COVID-19.

1.5. Comprehensive Objective

The primary objective of this project is to predict the severity of COVID-19 in patients by analyzing TCR sequences using a graph-based feature extraction method. The study aims to leverage the diversity and specificity of the CDR3 region in TCR sequences, using computational approaches like LZGraph to understand the immune response's structural and relational dynamics (Konstantinovsky and Yaari, 2023). By linking these features to clinical outcomes, the project seeks to provide actionable insights that can inform patient care and treatment strategies.

2. METHODS

2.1. Data Collection and Preprocessing

1) Data Source and Initial Loading:

The initial dataset was obtained from the ImmPort portal, focusing on a study titled "Severely ill COVID-19 patients display impaired exhaustion features in SARS-CoV-2-reactive CD8+ T cells" (PMID: 33478949, PMCID: PMC8101257, DOI: 10.1126/sciimmunol.abe4782) by Kusnadi et al., 2021. This dataset included clonotype IDs, CDR3 amino acid sequences, CDR3 nucleotide sequences, virus reactivity, clone sizes, and patient IDs.

The updated dataset was obtained from "Machine learning identifies T cell receptor repertoire signatures associated with COVID-19 severity" (PMID: 36670287, PMCID: PMC9853487, DOI: 10.1038/s42003-023-04447-4) by Park et al., 2023. This dataset included patient ID, CDR3 sequences for both alpha and beta chains, gene information, clonotype, disease condition, sex, age, and race.

2) Splitting CDR3 Sequences:

For the initial dataset, the CDR3 sequences were split into alpha and beta chains for both amino acid and nucleotide sequences. This separation was necessary to independently analyze the alpha and beta chain sequences and their potential impacts on disease severity.

3) Data Cleaning:

For both datasets, prefixes "TRA:" and "TRB:" were removed from the alpha and beta chain sequences to standardize the data and prepare it for further analysis. The updated dataset contained missing values in several columns, notably TRA_D_gene and TRB_D_gene, which were completely missing. Other columns with missing values included patient_subID, race, age, and d_cond.

4) Normalization and Clustering:

For the initial dataset, the 'Clone size' column was normalized using StandardScaler. K-means clustering with 8 clusters was applied to the normalized clone sizes to identify patterns and group similar sequences. The cluster distribution suggested potential outliers or less common patterns in the data.

5) Filtering and Renaming Columns:

For the updated dataset, the DataFrame was filtered to retain only the columns 'patient_ID', 'TRA_cdr3', and 'd_cond' (for alpha chains) and 'patient_ID', 'TRB_cdr3', and 'd_cond' (for beta chains). These were then renamed to 'patient_id', 'cdr3_amino_acid', and 'd_cond' respectively, focusing on both alpha and beta chains for subsequent analysis.

6) Exploration of Disease Condition:

The d_cond column, representing disease conditions, included categories such as 'mild', 'moderate', 'HD', and 'severe'. The counts of each category were analyzed, with 'mild' having the highest count followed by 'moderate', 'severe', and 'HD'. The updated dataset contained 270 unique patient IDs.

2.2. Feature Extraction and Graph Construction

1) Graph-Based Feature Extraction:

Beta sequences were extracted into a list, with None values filtered out. A dictionary for the LZGraph was generated using the generate_dictionary(6) function. The NaiveLZGraph from the LZGraph package was used to construct a graph from the beta sequences, where nodes represented unique sequences (clonotypes) and edges represented similarities or functional relationships.

2) Graph Analysis:

For the beta sequences, the initial graph contained 6075 nodes before removing isolates. After removing isolated nodes, the graph had 647 nodes. For the alpha sequences, the initial graph contained 29479 nodes before removing isolates. After removing isolated nodes, the graph had 846 nodes. Key metrics, such as eigenvector centrality, were calculated to measure the influence of each node within the network.

3) K1000 Diversity Index:

The K1000 Diversity index was computed to quantify the diversity of the TCR sequences, resulting in indices of 2152.77 for the initial dataset and 931.77 for the updated dataset. This index provides a measure of the immune repertoire's ability to recognize a wide array of antigens.

4) Graph Visualization:

The custom graph was converted to a NetworkX graph for visualization. Isolated nodes were removed, and a spring layout was used for positioning the nodes and edges. The visualization provided a clear depiction of the structural arrangement and connectivity of the TCR sequences.

5) Centrality Measures for a Specific Patient:

Data for specific patients, such as patient '100_1', were selected. The corresponding CDR3 amino acid sequences were used to create a graph using the NaiveLZGraph. Centrality measures (degree centrality, eigenvector centrality, betweenness centrality, and closeness centrality) were calculated for the nodes in the graph. These measures were organized into a DataFrame, which included a column for the patient ID. The final DataFrame included the centrality measures for 115 nodes associated with patient '100_1'.

6) Important Nodes Identification:

Eigenvector centrality and voterank were calculated to identify important nodes in the graph. The important nodes were mapped to their eigenvector centrality scores, and a structured array was created, containing the important nodes and their corresponding scores.

2.3. Comprehensive Analysis for All Patients in TCR beta and TCR alpha sequences

1) Iterative Analysis for All Patients:

A comprehensive analysis was performed for all patients by iterating over each patient in the filtered DataFrame. For each patient:

- The CDR3 amino acid sequences were extracted.
- A graph was created using the NaiveLZGraph for each patient's sequences.
- Isolated nodes were removed from the graph.
- Centrality measures (degree centrality, eigenvector centrality, betweenness centrality, and closeness centrality) were calculated for each node.
- Voterank was calculated to identify important nodes.
- A DataFrame was created for each patient containing the centrality measures and - voterank scores.
- Each patient's data was appended to a cumulative DataFrame named all_patients_voterank, which contains the centrality measures and voterank scores for all patients.

2) Aggregating Clinical Data:

The clinical data was aggregated to ensure a unique patient_id for merging purposes. The 'd_cond' column, representing disease condition, was assumed to be the same for each patient_id. The all_patients_voterank DataFrame was then merged with the aggregated clinical data, resulting in a merged DataFrame named merged_df_voterank with a shape of (30657, 8).

3) Data Cleaning:

The merged DataFrame had 19,046 rows with missing values. These rows were removed, resulting in a cleaned DataFrame named cleaned_df with a shape of (11,611, 8). The cleaned DataFrame was saved to a CSV file for further analysis.

4) Unique Nodes Analysis:

Unique nodes per disease condition (d_cond) were identified and analyzed for shared and unique nodes across classes. The analysis showed the number of unique nodes for each class and the number of nodes shared across all classes. Additionally, nodes unique to each class were identified.

5) Node Encoding and Feature Expansion:

A LabelEncoder was used to encode the 'node' column in the cleaned DataFrame, creating a new column 'node_encoded'. The DataFrame was then adjusted to maintain unique node features per patient by grouping and calculating the mean values for centrality measures. The DataFrame has pivoted to have 'node_encoded' as columns, resulting in a comprehensive feature set for each patient. This pivoted DataFrame was merged with the original patient data to include additional information like 'd_cond'.

2.4. Summary

This methodical approach, from data collection and preprocessing to feature extraction and graph construction, allowed for an in-depth analysis of TCR sequences. By leveraging advanced computational techniques and graph-based methods, the study aimed to uncover critical insights into the immune response and predict the severity of COVID-19 in patients. The updated dataset provided a broader foundation for understanding the complexity of TCR sequences, and subsequent analyses will build on these findings to enhance predictive models and clinical outcomes.

3. RESULTS

3.1.TCR beta sequences

3.1.1. RandomForestClassifier

For the RandomForestClassifier, the dataset was split into training and testing sets using an 80-20 split, ensuring that the target variable 'd_cond' was stratified. SMOTE (Synthetic Minority Over-sampling Technique) was applied to balance the class distribution in the training set. The parameter grid for RandomizedSearchCV included various values for max_depth, min_samples_split, min_samples_leaf, n_estimators, and class_weight.

RandomizedSearchCV was utilized to find the best hyperparameters, and the best model was evaluated on the test set. The classification report and confusion matrix for the best RandomForest model are presented below:

- Best Parameters: n_estimators: 200, min_samples_split: 2, min_samples_leaf: 1, max_depth: 20, class_weight: 'balanced_subsample'
- Best Accuracy (Cross-Validation): 0.744
- Test Accuracy: 0.48
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	1	16	5	0
moderate	0	10	6	3
severe	0	5	4	1

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.33	0.33	0.33	3
mild	0.53	0.73	0.62	22
moderate	0.39	0.37	0.38	19
severe	0.67	0.20	0.31	10
Accuracy			0.48	54
Macro avg	0.48	0.41	0.41	54
Weighted avg	0.50	0.48	0.46	54

3.1.2. BalancedRandomForestClassifier

The BalancedRandomForestClassifier was initialized with the calculated class weights to handle class imbalance. RandomizedSearchCV was used to optimize hyperparameters including max_depth, min_samples_split, min_samples_leaf, n_estimators, and class_weight. The best model was evaluated on the test set.

- Best Parameters: n_estimators: 100, min_samples_split: 10, min_samples_leaf: 2, max_depth: 30, class_weight: 'balanced'
- Best Accuracy (Cross-Validation): 0.372
- Test Accuracy: 0.39
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	5	0	0	0
mild	7	10	2	2
moderate	5	6	5	4
severe	4	1	2	1

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.24	1.00	0.38	5
mild	0.59	0.48	0.53	21
moderate	0.56	0.25	0.34	20
severe	0.14	0.12	0.13	8
Accuracy			0.39	54
Macro avg	0.38	0.46	0.35	54
Weighted avg	0.48	0.39	0.39	54

3.1.3. GradientBoostingClassifier

For the GradientBoostingClassifier, dataset was split and used RandomizedSearchCV to optimize the hyperparameters, including n_estimators, learning_rate, max_depth, min_samples_split, and min_samples_leaf. The best model was evaluated on the test set.

- Best Parameters: n_estimators: 200, min_samples_split: 5, min_samples_leaf: 1, max_depth: 5, learning_rate: 0.1
- Best Accuracy (Cross-Validation): 0.493
- Test Accuracy: 0.41
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	1	14	7	0
moderate	0	10	6	3
severe	0	5	4	1

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.50	0.33	0.40	3
mild	0.47	0.64	0.54	22
moderate	0.33	0.32	0.32	19
severe	0.25	0.10	0.14	10
Accuracy			0.41	54
Macro avg	0.39	0.35	0.35	54
Weighted avg	0.38	0.41	0.38	54

3.1.4. KNeighborsClassifier

The SMOTE was applied to balance the training data and used RandomizedSearchCV to optimize the hyperparameters for the KNeighborsClassifier. The parameter grid included n_neighbors, weights, algorithm, and p. The best model was evaluated on the test set.

- Best Parameters: weights: 'distance', p: 1, n_neighbors: 3, algorithm: 'ball_tree'
- Best Accuracy (Cross-Validation): 0.655

- Test Accuracy: 0.43
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	3	8	5	6
moderate	1	6	9	3
severe	1	0	4	5

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.17	0.33	0.22	3
mild	0.53	0.36	0.43	22
moderate	0.47	0.47	0.47	19
severe	0.36	0.50	0.42	10
Accuracy			0.43	54
Macro avg	0.38	0.42	0.39	54
Weighted avg	0.46	0.43	0.43	54

3.1.5. StackingClassifier

The StackingClassifier was constructed using KNeighbors and RandomForest as base models, with LogisticRegression as the meta-model. The model was fitted using SMOTE-enhanced training data and evaluated on the test set.

- Test Accuracy: 0.52
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	0	19	3	0

moderate	0	11	7	1
severe	0	1	7	2

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	1.00	0.33	0.50	3
mild	0.54	0.64	0.58	22
moderate	0.45	0.53	0.49	19
severe	0.60	0.30	0.40	10
Accuracy			0.52	54
Macro avg	0.65	0.45	0.49	54
Weighted avg	0.55	0.52	0.51	54

3.1.6. XGBClassifier

For the XGBClassifier, the dataset was split and used RandomizedSearchCV to optimize the hyperparameters, including n_estimators, learning_rate, max_depth, min_child_weight, subsample, colsample_bytree, and scale_pos_weight. The best model was evaluated on the test set.

- Best Parameters: subsample: 0.8, scale_pos_weight: 0.149, n_estimators: 100, min_child_weight: 1, max_depth: 10, learning_rate: 0.2, colsample_bytree: 0.6
- Best Accuracy (Cross-Validation): 0.456
- Test Accuracy: 0.54
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	0	19	3	0
moderate	0	11	7	1

severe	0	1	7	2

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	1.00	0.33	0.50	3
mild	0.59	0.86	0.70	22
moderate	0.39	0.37	0.38	19
severe	0.67	0.20	0.31	10
Accuracy			0.54	54
Macro avg	0.66	0.44	0.47	54
Weighted avg	0.56	0.54	0.50	54

3.2.TCR beta and TCR alpha sequences

<u>3.2.1 RandomForestClassifier</u>

- Best Parameters: n_estimators: 200, min_samples_split: 2, max_depth: 10
- Best Accuracy (Cross-Validation): 0.75
- Test Accuracy: 0.56
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	0	2	0	1
mild	1	20	1	0
moderate	1	10	7	1
severe	0	4	3	3

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*

HD	0.00	0.00	0.00	3
mild	0.56	0.91	0.69	22
moderate	0.64	0.37	0.47	19
severe	0.60	0.30	0.40	10
Accuracy				0.56
Macro avg	0.45	0.39	0.39	54
Weighted avg	0.56	0.56	0.52	54

3.2.2 BalancedRandomForestClassifier

- Cross-Validation Scores: [0.55, 0.65, 0.72, 0.73, 0.85]
- Mean Cross-Validation Score: 0.70
- Test Accuracy: 0.52
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	0	2	0	1
mild	1	18	3	0
moderate	1	10	7	1
severe	0	4	3	3

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.00	0.00	0.00	3
mild	0.53	0.82	0.64	22
moderate	0.54	0.37	0.44	19
severe	0.60	0.30	0.40	10
Accuracy				0.52

Macro avg	0.42	0.37	0.37	54
Weighted avg	0.52	0.52	0.49	54

<u>3.2.3 GradientBoostingClassifier</u>

- Best Parameters: n_estimators: 200, min_samples_split: 10, min_samples_leaf: 1, max_depth: 3, learning_rate: 0.2
- Best Accuracy (Cross-Validation): 0.7414
- Test Accuracy: 0.54
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	0	2	0	1
mild	1	18	3	0
moderate	1	10	7	1
severe	0	4	3	3

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.00	0.00	0.00	3
mild	0.56	0.64	0.60	22
moderate	0.52	0.63	0.57	19
severe	0.50	0.30	0.37	10
Accuracy				0.54
Macro avg	0.40	0.39	0.39	54
Weighted avg	0.50	0.54	0.51	54

<u>3.2.4 KNeighborsClassifier</u>

- Best Parameters: weights: 'distance', p: 1, n_neighbors: 3, algorithm: 'ball_tree'

- Best Accuracy (Cross-Validation): 0.6437
- Test Accuracy: 0.37
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	0	1
mild	4	5	9	4
moderate	2	11	5	1
severe	0	2	4	4

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.14	0.33	0.20	3
mild	0.50	0.18	0.27	22
moderate	0.55	0.58	0.56	19
severe	0.21	0.40	0.28	10
Accuracy				0.37
Macro avg	0.35	0.37	0.33	54
Weighted avg	0.44	0.37	0.37	54

3.2.5 StackingClassifier

- Base Models: KNeighborsClassifier(weights='distance', p=1, n_neighbors=3, algorithm='ball_tree'), RandomForestClassifier(n_estimators=100, random_state=42)
- Meta-Model: LogisticRegression
- Test Accuracy: 0.57
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*

HD	0	2	0	1
mild	0	18	3	1
moderate	0	7	10	2
severe	0	2	5	3

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.00	0.00	0.00	3
mild	0.62	0.82	0.71	22
moderate	0.56	0.53	0.54	19
severe	0.43	0.30	0.35	10
Accuracy				0.57
Macro avg	0.40	0.41	0.40	54
Weighted avg	0.53	0.57	0.54	54

3.2.6 KNeighborsClassifier with PCA

- Test Accuracy: 0.30
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	2	0	0	1
mild	7	7	4	4
moderate	1	9	6	3
severe	2	4	3	1

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	0.17	0.67	0.27	3

mild	0.35	0.32	0.33	22
moderate	0.46	0.32	0.37	19
severe	0.11	0.10	0.11	10
Accuracy				0.30
Macro avg	0.27	0.35	0.27	54
Weighted avg	0.33	0.30	0.30	54

<u>3.2.7 XGBClassifier</u>

- Best Parameters: subsample: 0.8, scale_pos_weight: 1.0, n_estimators: 100, min_child_weight: 1, max_depth: 5, learning_rate: 0.2, colsample_bytree: 0.8
- Best Accuracy (Cross-Validation): 0.7126
- Test Accuracy: 0.59
- Confusion Matrix:

	HD	*mild*	*moderate*	*severe*
HD	1	1	1	0
mild	0	19	3	0
moderate	0	11	7	1
severe	0	1	7	2

- Classification Report:

Class	*Precision*	*Recall*	*F1-Score*	*Support*
HD	1.00	0.33	0.50	3
mild	0.59	0.86	0.70	22
moderate	0.39	0.37	0.38	19
severe	0.67	0.20	0.31	10
Accuracy				0.54
Macro avg	0.66	0.44	0.47	54

Weighted avg	0.56	0.54	0.50	54

4. DISCUSSION

4.1 Analysis of Models on Beta Sequences Only

1) RandomForestClassifier:

The RandomForestClassifier achieved a test accuracy of 0.56, which indicates that it was moderately effective in classifying the disease severity based on the beta sequences. The model demonstrated high recall for the mild class, meaning it was proficient in identifying instances of mild disease. However, the model struggled significantly with the HD (healthy donor) and severe classes, where the recall values were very low. This suggests that while the RandomForest model was able to generalize well for the majority class (mild), it failed to capture the subtle distinguishing features necessary to accurately classify the minority classes. The model's stable cross-validation scores reinforce its reliability and robustness, but the significant performance drop for the minority classes indicates that further improvements are needed, particularly in handling imbalanced datasets.

2) BalancedRandomForestClassifier:

Designed to better handle imbalanced datasets, the BalancedRandomForestClassifier achieved a slightly lower test accuracy of 0.52 compared to the standard RandomForestClassifier. Despite its focus on balancing class weights, the model did not significantly outperform the standard version. However, it did show a slight improvement in recall for the severe class, indicating its utility in identifying minority class instances. The marginal overall improvement suggests that while balancing techniques help, the inherent complexity and variability within the data might require more advanced methods or additional data preprocessing to achieve significant performance gains.

3) GradientBoostingClassifier:

The GradientBoostingClassifier achieved a test accuracy of 0.54, similar to the RandomForestClassifier. This model demonstrated better precision for the moderate class but struggled with recall for the severe cases, reflecting its inconsistency in handling different classes. The macro and weighted averages of precision and recall indicate that the model's performance was not uniform across all classes, highlighting areas for improvement. The GradientBoostingClassifier's ability to capture complex patterns is evident, but it might require further tuning or enhanced feature engineering to improve performance on the minority classes. The stable cross-validation scores suggest that the model generalizes well but might not be the optimal choice without further refinement.

4) KNeighborsClassifier:

The KNeighborsClassifier (KNN) displayed the lowest performance among all models, with a test accuracy of 0.37. The model had low precision and recall across all classes, particularly struggling with the mild and severe categories. This poor performance suggests that KNN's reliance on local information and distance metrics is insufficient for the dataset's complexity. The inability to capture the intricate relationships within the data indicates that KNN might not be suitable for this type of classification task, where more sophisticated approaches are needed to handle the inherent variability and complexity.

5) StackingClassifier:

The stacking classifier, which combines KNN and RandomForest with Logistic Regression as the meta-model, achieved a test accuracy of 0.57. This model significantly improved recall for the mild class but continued to struggle with the HD class, where precision and recall were zero. The improved performance in identifying mild cases demonstrates the effectiveness of combining different algorithms to leverage their strengths. However, the persistent difficulty with the HD class indicates that even a sophisticated ensembling approach might require further refinement or additional data preprocessing to handle certain minority classes effectively.

6) KNeighborsClassifier with PCA:

Applying PCA to reduce dimensionality did not improve the performance of KNN, resulting in a lower test accuracy of 0.30. This outcome suggests that dimensionality reduction might not effectively capture the essential features needed for KNN to perform well on this dataset. The potential loss of crucial information during PCA might have contributed to the model's poor performance, highlighting the importance of retaining key features for effective classification. This result underscores the limitations of using KNN with PCA for this specific dataset, where preserving detailed information is critical for accurate classification.

7) XGBClassifier:

The XGBClassifier achieved the highest test accuracy of 0.59, indicating its robustness in handling complex datasets. The model performed well across all classes, especially in identifying mild and moderate cases, with relatively balanced precision and recall. The model's ability to handle class imbalance effectively, combined with its robust optimization techniques, likely contributed to its superior performance. The consistent cross-validation scores further underscore the model's reliability and robustness, making it the most effective model among those tested for this dataset.

4.2. Analysis of Models on Combined Alpha and Beta Sequences

1) RandomForestClassifier:

With the combined sequences, the RandomForestClassifier maintained a test accuracy of 0.56, similar to its performance in Part 1. The model continued to show high recall for the mild class but struggled with precision for the HD and severe classes. This consistency suggests that the addition of alpha sequences did not significantly enhance the model's performance, indicating that the beta sequences alone might capture sufficient information for the RandomForestClassifier. The model's stable performance across both parts highlights its robustness but also points to a need for further improvement in handling minority classes.

2) BalancedRandomForestClassifier:

The balanced approach did not significantly improve performance over the standard RandomForestClassifier, maintaining a similar test accuracy of 0.52. The model performed slightly better in identifying severe cases, indicating its effectiveness in handling class imbalance. However, the overall improvement was marginal, suggesting that the combined sequences might not have provided additional discriminative features for the balanced model. This result highlights the need for more advanced techniques to effectively leverage the additional information from the combined sequences.

3) GradientBoostingClassifier:

The GradientBoostingClassifier showed consistent performance with a test accuracy of 0.54. The model's ability to generalize well across different splits suggests its robustness, but it also highlights its limitations in improving performance for minority classes. The addition of alpha sequences did not significantly enhance the model's discriminative power, indicating that further tuning or feature engineering might be necessary. The consistent performance across both parts suggests that while GradientBoosting is effective, it may require additional enhancements to handle the full complexity of the combined data.

4) KNeighborsClassifier:

The KNeighborsClassifier continued to show the lowest performance among all models, indicating that the combined sequences did not enhance its capability to distinguish between classes effectively. The model's reliance on local data points might be insufficient for capturing the complex relationships in the dataset, resulting in consistently poor performance. This outcome reinforces the limitations of KNN for this specific task and suggests that more sophisticated methods are needed to handle the data's complexity.

5) StackingClassifier:

The stacking classifier showed improved performance compared to individual models, achieving a test accuracy of 0.57. This result suggests that combining multiple algorithms can leverage their strengths, leading to better generalization. However, the persistent difficulty with the HD class indicates a need for further refinement in the stacking approach. The consistent performance improvement highlights the potential of ensembling methods but also underscores the challenges in handling certain minority classes effectively.

6) KNeighborsClassifier with PCA:

PCA did not improve the performance of KNN, resulting in a lower test accuracy of 0.30. This outcome suggests that dimensionality reduction might not effectively capture the essential features needed for KNN to perform well on this dataset. The potential loss of crucial information during PCA might have contributed to the model's poor performance, highlighting the importance of retaining key features for effective classification. This result underscores the limitations of using KNN with PCA for this specific dataset, where preserving detailed information is critical for accurate classification.

7) XGBClassifier:

The XGBClassifier continued to achieve the highest test accuracy of 0.59, indicating its robustness in handling complex datasets. The model's ability to effectively manage class imbalance and optimize multiple parameters likely contributed to its superior performance. The consistent performance across both parts suggests that XGBClassifier can effectively leverage the additional information provided by the alpha sequences, making it the most effective model among those tested for this dataset.

Conclusion

Across both parts of the analysis, the XGBClassifier consistently outperformed other models, demonstrating superior accuracy and robustness. The RandomForestClassifier and GradientBoostingClassifier showed moderate performance, with the stacking classifier leveraging multiple algorithms to achieve better generalization. The KNeighborsClassifier performed poorly, indicating its unsuitability for the dataset's complexity. The addition of alpha sequences did not significantly enhance the performance of most models, suggesting that beta sequences alone might capture sufficient information for effective classification.

The consistent performance of XGBClassifier highlights its robustness in handling complex datasets and effectively managing class imbalance. The stability of the RandomForestClassifier and GradientBoostingClassifier across both parts suggests that while these models are effective, they might require further refinement to handle the full complexity of the data. The poor performance of KNeighborsClassifier, even with PCA, underscores the limitations of using distance-based methods for this specific task.

Future Work

To further enhance model performance, future work could explore the following avenues:

- Feature Engineering: Investigate additional feature engineering techniques to extract more discriminative features from the data. This could include domain-specific knowledge to identify key patterns in the sequences.
- Advanced Algorithms: Explore advanced machine learning algorithms and deep learning techniques, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), to capture complex patterns in the sequence data.
- Data Augmentation: Implement data augmentation techniques to increase the diversity of the training data and improve the model's generalization capabilities.
- Model Ensembling: Investigate more sophisticated ensembling techniques, such as gradient boosting with different base learners, to leverage the strengths of various models.
- Hyperparameter Tuning: Conduct more extensive hyperparameter tuning, potentially using Bayesian optimization or other advanced search methods, to find the optimal settings for each model.
- Cross-Validation: Employ more robust cross-validation strategies, such as nested cross-validation, to better estimate model performance and avoid overfitting.
- Class Imbalance: Continue to address class imbalance through techniques like SMOTE, ADASYN, or cost-sensitive learning to improve model performance on minority classes.
- External Validation: Validate the models on external datasets to ensure their generalizability and robustness in different settings.
- Utilize AAPLZGraph and Clinical Data: Integrate AAPLZGraph along with genetic and other clinical data to enrich the feature set and potentially

References

(26) Extracting Node Level Features from Graph Networks for Machine Learning Models : Part 2 of X of my notes | LinkedIn [WWW Document], n.d. URL https://www.linkedin.com/pulse/extracting-node-level-features-from-graph-networks-machine-taneja-ssmgf/ (accessed 5.8.24).

de Greef, P.C., Lanfermeijer, J., Hendriks, M., Cevirgel, A., Vos, M., Borghans, J.A.M., van Baarle, D., de Boer, R.J., 2023. On the feasibility of using TCR sequencing to follow a vaccination response – lessons learned. Front Immunol 14, 1210168. https://doi.org/10.3389/fimmu.2023.1210168

Grando, F., Granville, L.Z., Lamb, L.C., 2019. Machine Learning in Network Centrality Measures: Tutorial and Outlook. ACM Comput. Surv. 51, 1–32. https://doi.org/10.1145/3237192

Katayama, Y., Yokota, R., Akiyama, T., Kobayashi, T.J., 2022. Machine Learning Approaches to TCR Repertoire Analysis. Front. Immunol. 13. https://doi.org/10.3389/fimmu.2022.858057

Konstantinovsky, T., Yaari, G., 2023. A Novel Approach to T-Cell Receptor Beta Chain (TCRB) Repertoire Encoding Using Lossless String Compression. Bioinformatics (Oxford, England) 39. https://doi.org/10.1093/bioinformatics/btad426

Kusnadi, A., Ramírez-Suástegui, C., Fajardo, V., Chee, S.J., Meckiff, B.J., Simon, H., Pelosi, E., Seumois, G., Ay, F., Vijayanand, P., Ottensmeier, C.H., 2021. Severely ill COVID-19 patients display impaired exhaustion features in SARS-CoV-2-reactive CD8+ T cells. Sci Immunol 6, eabe4782. https://doi.org/10.1126/sciimmunol.abe4782

Li, Y., Yang, X., Zhang, X., Xi, M., Lai, X., 2022. An improved voterank algorithm to identifying a set of influential spreaders in complex networks. Front. Phys. 10. https://doi.org/10.3389/fphy.2022.955727

Park, J.J., Lee, K.A.V., Lam, S.Z., Moon, K.S., Fang, Z., Chen, S., 2023. Machine learning identifies T cell receptor repertoire signatures associated with COVID-19 severity. Communications Biology 6, 76. https://doi.org/10.1038/s42003-023-04447-4