CS155 Group Project 1 Report

Halle Blend, Georgia Malueg, Rachael Kim, Jayden Nyamiaka

February 2023

1 Introduction [15 points]

1. **Group Members:**
   Halle Blend, Georgia Malueg, Rachael Kim, Jayden Nyamiaka
2. **Kaggle team name:** Ragecage
3. **Ranking on private leaderboard: 18th**
4. F_2 **score on the private leaderboard: 0.85447**
5. **Colab Link:**
   Motility Features
   Model
6. **Piazza Link:** Ragecage
7. **Division of Labor:**
   Halle: Worked on preprocessing the data. Worked on designing/hypertuning XGBoost Models. Also worked on adding additional motility features.
   Georgia: Worked on optimizing the XGBoost Models by hypertuning the parameters. Also worked on decreasing overfitting and changing the XGBoost model parameters.
   Rachael: Worked on researching and adding additional motility features.
   Jayden: Worked on scoring the models. Also worked on validation techniques.

2 Overview [15 points]

Models and techniques tried

We dedicated most of our time fine-tuning the model we developed. We only tried two models (Linear Regression and XGBoost), and XGBoost happened to work really well with this dataset, so for rest of the week, we worked on feature engineering, optimization of the model and development of validation/regularization methods to reduce the overfitting in our model.

For feature engineering, we looked into multiple research done on motility in the context of machine learning. We mainly looked at three research papers (listed in the approach section) that explained feature extractions from motility data. We then adapted those to our model and tested them out to see if they improve accuracy. Details of feature engineering are explained in the Approach section.

For optimization of hyperparameters, we mainly tried out two methods: grid search and `hyperopt`. We started off with grid search, but we decided to use Bayesian optimization through the hyperopt package, which works faster than grid search. And, for more general model optimization purposes, we considered weighting our features to optimize the model for our dataset. We will explain our model optimization methods in further detail in Model Selection section.

We mainly used K-Fold Cross validation method for our validation and regularization. We noticed that there was an overfitting in our model, so we constructed our final model by gathering predictions made by each model in our K-Fold validation and averaging them out to produce our final predictions. To reduce overfitting, we also slightly changed the hyperparameters of XGBoost so the model stops creating nodes earlier.

Work timeline

Features	Wed (2/8)	Thur (2/9)	Fri (2/10)	Sat (2/11)	Sun (2/12)	Mon (2/13)	Tue (2/14)
Setup	Setting up the project (Creating a Kaggle team, Google Colab notebooks and Starting Report)						
Model Development/ Optimization		Researching and Developing preliminary XGBoost model (Tried out Linear Regression and XGBoost)			Hyperparameter optimization: using `hyperopt` to find the optimal hyperparameters for our model		Testing our hyperparameters for the last time
Feature Engineering			standard deviation in turn angle	Tried additional features such as progressivity, mean signed turning angle, track length with sliding window		Testing features	
Data Processing				Normalization using Min-Max scaler			
Validation and Regularization				Developing validation methods using K-Fold	Adding regularization methods to tackle overfitting - adding loss function, averaging our model predictions from K-Fold, turning parameters, etc.		
Kaggle Submissions			First Submission			Second Submissions	Final Submissions

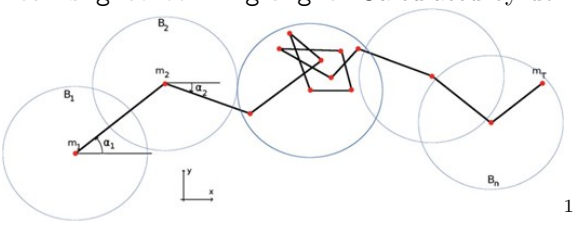
3 Approach [20 points]

Data exploration, processing and manipulation

Feature Engineering
We attempted to add a few features to the set of features given in the code example, but not all of them ended up being successful. These features were implemented based on the research that had been previously conducted on motility: motilitAI , Heteromotility, Study on single-cell motility.

Here are the new features we have attempted:

1. `angle_stddev`: the standard deviation in the angles between neighboring points.
2. `track_length_sliding_window_5`: the total track length with the sliding window of 5. We calculated the sum of the Euclidean distance between i-th point and (i+5)-th point, for every point in the array.
3. `progressivity`: the progressivity of the particle, calculated via track length divided by e2e distance.
4. `mean_signed_turning_angle`: Calculated by doing: $\alpha_2 - \alpha_1$ (α_2, α_1 shown in the diagram below).



[1]

We decided to take out `progressivity` and `mean_signed_turning_angle` at the end, because we realized these features were actually decreasing our accuracy by a lot (nearly .1 difference in F_2 score). We did this by cycling through features and comparing the accuracies of our model with each feature removed using our accuracy measurements. We also confirmed this through trial and error and a few submissions. There could be multiple reasons as to why this has happened, but one of the most relevant reasons would probably be the correlation between these two new features and the other features. Since these two features calculate their values based on other features, they would likely be strongly correlated, and thus create a bias in our model. Another possibility is that these features simply may not have a strong correlation in determining motility.

Data Preprocessing
After we have processed our train and test data using our features, we normalized the data using min-max scaling. Normalization didn't improve our model's F_2 score, but we still kept it for consistency in our data. We also shuffled our data.

Details of models and techniques

We initially used Linear Regression to start off the project, but quickly realized linear regression would probably not be the best model, since our features are not necessarily linearly correlated to the motility of the particle. We verified this by training a linear regression model on our train data, which returned relatively low accuracy. We then decided to try the XGBoost model because one of us came across XGBoost before and thought that an implementation of gradient boosted decision trees would likely be the best model types for the dataset. The benefit of the model was clear: the model had very high F_2 score once submitted to Kaggle. Since we had intuition that the complexity of the model matched the complexity of the dataset really well (both very complex), it gave us a strong reason to stick with XGBoost. One of the disadvantages of the model was overfitting. Both the training accuracy and K-Fold validation test accuracy came out to .99. We tried hyperparameters we got from hyperopt package, but regardless of what hyperparameter we used, we still had issues with overfitting. Since our model was outputting 0.99 test validation even with K-Fold Cross Validation, this gave us intuition that the distribution of our train data was different than the distribution of the test data that we needed to predict. We didn't have much time to look into this, but we believe it to be a significant reason why our model's performance was not higher. We implemented some strategies to tackle this, which will be explained in the next section.

[1] https://academic.oup.com/bioinformatics/article/31/12/i320/215036

4 Model Selection [20 points]

Model Optimization Objectives

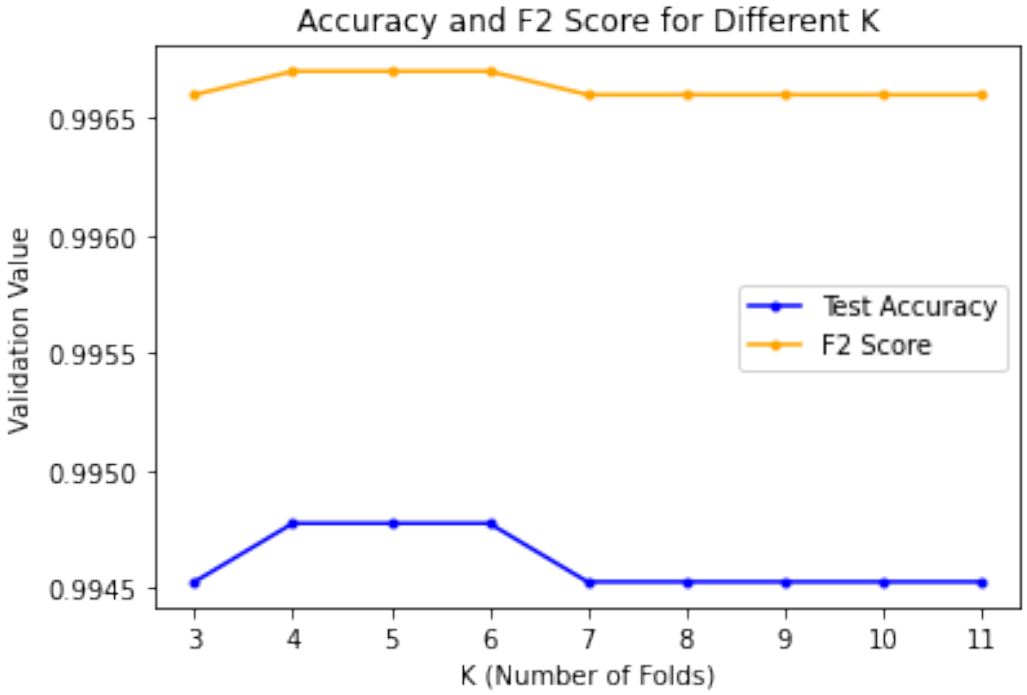
There are two main strategies we attempted to optimize our model. First, we tried feature weighting. We looked at the set of features we had and tried to figure out which feature could be more relevant and correlated to motility. However, we soon realized that figuring out this correlation is difficult, especially with the lack of understanding of science behind motility. So, considering the potential risk of bias and overfitting, we decided not to implement feature weighting in our model optimization process.
Our second strategy was to use `hyperopt` package to find the best hyperparameters for our XGBoost model. `Hyperopt` is a hyperparameter optimization method which uses a form of Bayesian optimization for paramater tuning. By randomly selecting within a specified range for each hyperparamater, we used `hyperopt` to generate 100 different models with different hyperparameters. Then, our code stored the hyperparameters of the model with the best validation score. However, we couldn't really utilize `hyperopt` effectively in our model due to the overfitting the `hyperopt` model created. Our XGBoost model was already slightly overfitted to our train data, and adding hyperparameters made it worse by reducing the test accuracy even further to .74 while increasing the validation accuracy to .99. We were overly tuning the parameters to the training data, by picking parameters that model the train data well. And this causes a big overfitting problem in our model. So, we attempted to reduce overfitting by reducing the max tree depth and increasing the regularization parameters. However, the tuned model's accuracy still remained lower than with default hyperparameters. And we ran out of time by the end of the project, so we decided to take out the hyperparameter section of our code from our final submission.
We knew the binary logistic loss function ws typically used for classification problems with XGBoost, so we decided to try it out. It proved to perform well, resulting in a jump in accuracy, so we stuck with it.

Scoring

We didn't have to compare different regression models because our XGBoost Classification model, which was one of the first models we tried, happened to work very well. So, we decided to develop our model through optimization methods explained above. Thus, we didn't have to perform scoring between different models, but we did do scoring between different chosen hyperparameters. We did test the accuracy of our model after each optimization, however. We initially compared the accuracy by submitting our model predictions to Kaggle - which wasn't the ideal way of scoring the models. So, we developed certain validation and test strategies to check the performance of our model. First, we simply developed an accuracy metric. It took the correct labels and predictions and computed the ratio of correct to total. Secondly, we used F_2 Score from Python's sklearn module (since that was the metric used for leaderboard scoring) to test the our models. We used these two metrics throughout testing any and all of the models we implemented.

Validation and Test

We used K-Fold Cross validation strategy to validate our models. We knew from lecture that K-Fold Cross Validation works well for both validating model performance and regularizing the model to prevent overfitting. We were having trouble with both of these, so K-Fold Validation mitigated both of our problems. Thus, we decided to test out different K fold splits between 3 and 11 and see which K yielded the best model performance. To do this, we split our training data: 75% training and 25% testing. From there, we performed K-Fold Cross Validation on the 75% training data and used the 25% testing data to calculate the F_2 score and accuracy for each K. We computed predictions by averaging the predictions for each point for all K models. Doing this analysis, we saw that the test accuracy and F_2 score were highest for folds 4, 5, and 6, so we chose 5 folds. The graph for this comparison can be seen below. The accuracy and F_2 score were only slightly higher by ¡ 5 more points predicted correctly, showing that it probably didn't matter which K we chose. Using 5-Fold Cross Fold Validation to predict the test data, our F_2 score on the leaderboard went up by about 2%. This demonstrates the benefit that Cross Validation added. It also supported our hypothesis that our models are overfitting because Cross Validation is known to regularize data. So, cross validation helped mitigate our overfitting. In doing Cross Validation, our F_2 score was super high at ¿99%, yet our performance on Kaggle was only 85%. More than overfitting, we believe this meant there was a different in the distributions between the train data and the test data. This meant we needed to put more time into examining the input data, seeing the difference between the distributions, so we could test on different distributions and see which performed better on the actual test data. If given more time, we would have done this.


Accuracy and F2 Score for Different K
Validation Value
0.9965
0.9960
0.9955
0.9950
0.9945
3
4
5
6
7
8
9
10
11
K (Number of Folds)
Test Accuracy
F2 Score

5 Conclusion [20 points]

Insights

Here is the complete list of features that we have tested for our model: `mean_step_speed`, `stddev_step_speed`, `track_length`, `e2e_distance`, `duration`, `angle_stddev`, `track_length_sliding_window_5`, `track_length_sliding_window_10`, `progressivity`, `mean_signed_turning_angle`. And out of these features, we have included the following: `mean_step_speed`, `stddev_step_speed`, `track_length`, `e2e_distance`, `duration`, `angle_stddev`, `track_length_sliding_window_5`.
We did a bit of analysis on our feature train data to see if there is any correlation between any of our features and the label. We first looked at the linear correlation, which showed us that `angle_stddev` and `mean_step_speed` are most linearly correlated to our label values.However, when we looked at individual feature, we realized that it is impossible to use any of these features as linear classifiers for our labels. So, it is really hard to say that one feature has overpowering impact on our model's classification algorithm than other features.
However, logically approaching this problem, we believe that `mean_step_speed` could be the feature that influenced our predictions the most. Since the duration is not equal for all particles, for us to really understand how much each particle travelled for a given unit of time, we used `mean_step_speed` for better insight.
If we were to rank our features in the order of importance, we would rank it in the following order: `mean_step_speed`, `track_length_sliding_window_5`, `stddev_step_speed`, `track_length`,`angle_stddev`, `e2e_distance`, `duration`. We ranked features that have relevance to physical motions first. `e2e_distance` has relatively less significance than `track_length` because it doesn't accurately represent paths of the particles that may have a lot of turns. `track_length` is more accurate representation of the total motion of the particle. The features that have been listed in the rank are the features that were determined to have positive impacts on our model. They improved our F_2 score when we included these features in our feature training.
Features we have not included - `track_length_sliding_window_10`, `progressivity`, `mean_signed_turning_angle` - have the least significance. These features had negative influence on our model accuracy, so we ended up taking these out. These features were either irrelevant to our model or too closely correlated to other features. `progressivity` was pretty irrelevant to our model, because it is not the most accurate measure of the motion, although it may give us insight on whether a particle was moving in one direction or not. `track_length_sliding_window_10` and `mean_signed_turning_angle` were too closely correlated to other features we used, such as `track_length_sliding_window_5` and `angle_stddev`, that it introduced bias to our model and made our model overfit to certain features more than other ones.
Overall, we learnt the importance of feature engineering from this project. Developing relevant features can really boost up the accuracy of the model, but including one harmful feature (that is either irrelevant or too biased) could lead to a huge drop in the accuracy. People assume that machine would do all the work if we simply feed the data, but we realized that it is important to feed good data to our model to get meaningful result out of it. We also got to test out various validation and optimization strategies. This project was harder than we expected, because unlike our previous assignments, we had to predict labels for test set that did not have labels. Thus, we had to be careful that we don't overfit to our train data and develop validation strategies to make sure we can prevent overfitting.

Challenges

One of the initial challenges was data processing. Part of our data was getting lost, and this lowered our accuracy as we were simply not making predictions for some of the particles. However, we were able to fix this error by debugging our code and rewriting our data processing codes. Next project, we could potentially allocate more time for data processing, since using clean data is important for model accuracy.
Our biggest challenge was tackling overfitting. Since we trained our model on the train data alone, our model tended to overfit to our train data. And it took us time to figure out the cause of overfitting and develop validation and regularization methods to prevent this. We tried to improve it by reducing the training accuracy, but some of the regularization methods we tried (e.g. adding in hyperparameters that we thought would reduce overfitting) would end up making it worse by reducing our test accuracy even further, often because we were making our model worse by introducing excessive regularization. And as aforementioned, this regularization step was particularly challenging in this project, because unlike the previous projects, we had to deal with test data with no labels. Towards the end, we realized the discrepancy in accuracy was probably due to the differences in the distribution between the test and train data; however, we did not have much time to look into this. If we had time, we would have plotted the differences in the feature distribution to find which points we should filter out, resulting in a better distribution and thus a better model.

6 Extra Credit [5 points]

Why F_2

The F2-measure places more emphasis on recall than precision. The models were evaluated using the F2-measure because we cared more about recall which in this project was identifying as many particle tracks that are predicted to have motility-based bio-signatures. Placing more emphasis on recall is important because false negatives are worse than false positives in this scenario. Furthermore, the F2 evaluation metric is suitable for imbalanced, real-world data sets where the number of positive samples is much smaller than the number of negative samples. So, the F2-measure is more appropriate than the F1-measure because the F2-measure takes into account both the number of true positives and the number of false negatives. Therefore, the F2 evaluation metric was a more appropriate choice than the F1 evaluation metric because we want to identify the most particle tracks that have motility-based bio-signatures and we are using a real-world data set with less far less positive samples than negative samples.

Parallelization

Hyperopt is a python package used for serial and parallel optimization. While Hyperopt did not contribute to our final model, we did attempt parralelization. Furthermore, we could have parallelized the cross-validation score.

Additional Data

The data we had for this challenge only provided (x,y) coordinates for each t value. However, it did not provide any information on the data collection setting. A presence of external force can totally change our interpretation of data, since external force may be pushing the particle around and giving us misleading information in terms of track length or mean step speed. When we looked into the data, this actually happened to be true, because none of the features could linearly classify our data points even though some of the features are directly related to motility. A particle that moved 350 units over given time with 6 units/time of mean step speed, for instance, had no motility. But we were given no data that we can use to explain as to why a particle with such movement had no motility, while another particle with a similar trajectory did. So, if we are given extra data on external forces, we believe we can improve our model accuracy by a lot.

We believe that it could have helped us to have a better insight into the distribution of test data, because part of the reason why overfitting occurred in our model was because the train data did not accurately represent the whole dataset and had a different distribution from the test data. So, when we trained our model on train data, it couldn't predict test data well.

New Features

We implemented a few new features in order to better predict the motility classification. One new feature was the standard deviation of the angles between points of the entire track. Another new feature was the length of the entire track with the window size 5. Additionally, we added the progressivity of particle represented by the length of the track divided by the end-to-end distance. Lastly, we added mean signed turning angle. These features helped the model to predict the motility and increase our accuracy.