1 Introduction [15 points]

- Group members: Michelle Li, Manuel Rodriguez, Kevin Do, Alicia Zhang
- Kaggle team name: Sir Cumference
- Ranking on the private leaderboard: 45
- F_2 score on the private leaderboard: 0.78189
- Colab link:
  https://colab.research.google.com/drive/1649hQ6ClNqS7PsHCJrWY9c1pR7uRZBpN
- Piazza link: https://piazza.com/class/lbv0docn6037fw/post/368
- Division of labor:
  Michelle- model structure (MLP, RNN), debugging features, OH
  Manuel- model (XGBoost, DT)/feature work (avg_accel, avg_turn_angle)
  Kevin- model structure (XGBoost, DT), OH
  Alicia- feature work (linearity, turn freq, pure residuals), OH

2 Overview [15 points]

Models and techniques attempted

1. RNNS: At first, we wanted to use the sequential properties of our data to incorporate RNNs as a feature. We wanted to train an RNN for every track and see if the last time step (or last few time steps) were predictable. If they were easily predictable, then the cell most likely isn't alive, as it can be easier to model. This ended up requiring a lot of additional work, and as we already had a working decision tree model, we decided to focus on engineering other features instead.

2. XGBoost: After talking to some TAs, we also decided to attempt XGBoost and MLPs. However, we realized that no matter what features we put in, only one feature would be heavily emphasized. We found out by looking at the feature weights, only one of them would have a really high weight, and all others would have nearly 0. This didn't make sense as we knew that the features we worked on should logically be helping the model make predictions, and it was a problem with how XGBoost was deciding its splittings.

3. MLPs: While we were working on XGBoost, we also decided to try implementing the MLP models to see how they compared. MLPs are a bit more difficult because they require parameter tuning like NNs. We used GridSearchCV but it still took a lot of time to find which parameters worked best because we'd need to train all of them. After running through variations max iterations, alphas, hidden layer sizes, and activation types, our best model predicted an accuracy of 73% on the given features. We ended up just sticking to decision trees as they're easier and gave better results.

4. Decision Trees: We started off with sklearn's random forests, and we ended up having to switch back to them from XGBoost. This was our final attempt at trying to get a valid weighting of our features. After improving and implementing our features, we wanted to at least make sure the model saw their importance. By using sklearn's random forest, we found a model that weighted the features accordingly. This helped us move back to implementing some last features that should help and running our updated model to create our final submission.

Work Timeline

First, we developed an initial model with the Random Forest. From there, we developed an XGBoost model, and found its optimized parameters with CVGridSearch. We started feature engineering by developing max_acceleration, turn_frequency, and by splitting the data into 3 sections, we obtained features for the first, second and third parts of the time series. However, the new features were not contributing towards accuracy because the model was not weighting the new features at all, so we went through a whole debugging process. We ended up removing the splitting of the data then implemented the linearity feature using sklearn's linear regression and R^2 methods. However, our model was still not showing any influence from the features, even though our plots showed that our features were useful. Eventually, we switched back from XGBoost to sklearn's Random Forest, which was able to properly use our features. This resulted in our highest accuracy yet. We then implemented turn angle and pure residual measure features, but those only contributed a negligible percent of increased accuracy.

3 Approach [20 points]

Data exploration, processing and manipulation

Ideas extracted from reading paper 1

1. "Individual cells were tracked, and velocities, accelerations, turn frequency,... classifying tracks into "motile" vs. "non-motile" based upon a defined algorithm."
   (a) We implemented avg velocity, avg acceleration, and turn frequency
2. "increasing numbers of immobile cells collected on the sample chamber surface as temperatures increased"
   (a) Could see if cell hasn't moved in N time steps, i.e. immobile (change in displacement)
3. "Dead cells showed little movement until temperatures >60°C, at which point convection currents became appreciable; however, these were readily distinguished from active motility by inspection as well as automated tracking tools."
   (a) Could also see if acceleration is 0 because that could mean that the cell is dead and is just being moved by convection currents.
4. "With the spatiotemporal points, HELM computes approximately two dozen metrics that form a feature vector for each track. These include features like mean speed, mean turn angle, total track displacement, etc."
   (a) This confirms the importance of some of the features we were given, for example avg speed. We also went on to implement duration and linearity as features. Where linearity uses R^2 to show deviations from the expected track route (used linear regression).

Other ideas:

1. Second-order differences: Calculate the difference between consecutive data points, and then calculate the difference between those differences. Identifies if the data is changing at an accelerating or decelerating rate.
2. Seasonal decomposition: Decompose the data into its trend, seasonality, and residual components.
3. Autoregression: Model the time-series data as a function of past values to create autoregressive features. Checks if the data is influenced by past values.
   (a) This could be done using an RNN, predicting the next position in the data. If there is a lot of uncertainty, it might be the case that the particle's movement cannot be predicted because it is alive rather than being influenced purely by physical laws and its surroundings.
4. Breaking up the existing starting features into smaller chunks (first third, second third, and last third of the points) to give more information.

4 Model Selection [20 points]

Notebooks: [Final Model Used] — [Feature Generator]

Scoring

For the scoring metric, we first used the validation accuracy given to us by 10-fold cross validation to determine which parameters would best train our Random Forest model. After finding the best set of hyperparamters, we train a model with those hyperparameters but now using the entire training set, so we are now no longer using a validation test set.

Firstly, we assumed that the optimal number of classifiers should be around 126, because in Lecture we discovered that the number of classifiers should be around the square root of the number of training data values (we have around 16000 data points so $\sqrt{16000} \approx 126$). We then continued to further finetune the parameters in an attempt to increase the test accuracy.

Later, we implemented the F_2 metric that was described in the problem, as that's the loss that we're trying to optimize. The best hyperparameters were found using CV gridsearch, and the model with the best validation accuracy was then retrained on the entire dataset. We also tried using XGBoost, but then we began having to see an issue in which models using XGBoost and GridSearch overly weighed the importance of the Mean Step Speed feature (around 80-90%). Even when including other features that seemingly should be very relevant like the Mean Squared Error and the Mean Step Acceleration features, the XGBoost model would always weigh the mean step speed feature with an importance $\geq 80\%$.

Testing by removing the mean step speed from the dataset, the next feature, the Standard Deviation of Speed, would now be overly represented. We believe this phenomenon was caused by large amounts of overfitting by the XGBoost + GridSearch model. Due to this, we switched back to a normal Random Forest model, and did a gridsearch on the 'min leaf size' and 'number of classifiers' hyperparameters. With this we found that the best classifier had 'min leaf size' equal 9 and 'number of classifiers' equal 100.

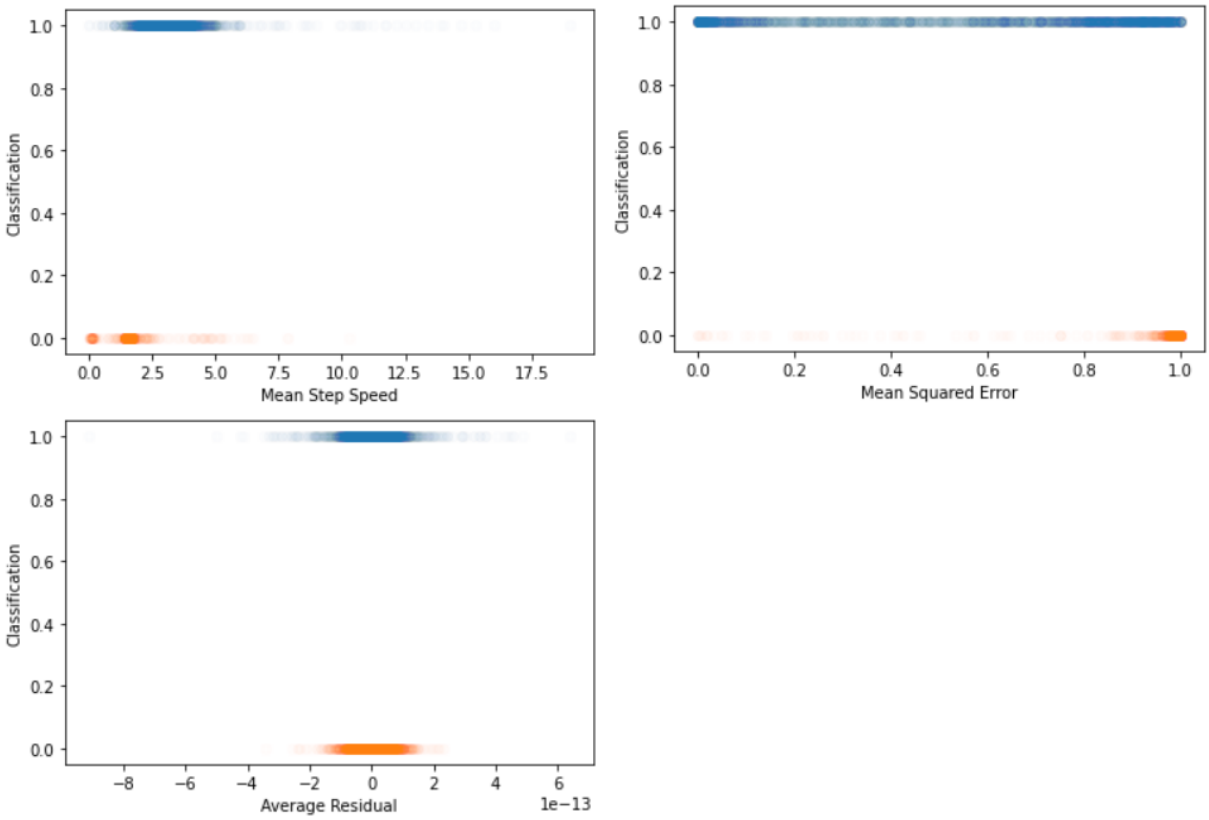
Validation and test

Firstly, we just used 10-fold cross validation to see how good the model was, and just used the best estimators. To see what hyperparameters of the model would perform best, we just used a gridsearch. Luckily for us, sklearn just has CV gridsearch, so it would tell us what the best model parameters were, based off of the scores for cross validation. There isn't really anything special about this, as we figured that CV gridsearch could do everything for us with the correct scoring measure.

5 Conclusion [20 points]

Insights

We noted that the most important features were the ones dealing with the linearity of the dataset. This can be seen from the track visualization example code in which it is clear that in non-motile tracks, motion looks very linear, whereas in motile tracks, there is much more variation and less clear of a pattern (though usually still somewhat linear). A general trend is that all the motile tracks had a greater standard deviation over their features than the non-motile tracks. Here we show plots for Average Step Speed, Average Mean Squared Error from Line of Best Fit, and Average Residual.



Challenges

An obstacle we encountered was building a model that properly weighted our features. Much of the project was spent on testing different models and tweaking their parameters in order to find the best performance. We realized that our initial model was causing issues when our implemented features were not significantly improving our accuracy. Despite engineering more features, they would not contribute towards our test accuracy unless we changed our model. Thus, our final model ended up being a Random Forest Classifier once again.

6 Extra Credit [5 points]

1. We can see that false positives and false negatives are a thing, so if we're just using the standard number of samples classified correctly / number of sample classified wrong, we can actually try training some really bad model somehow and just flip the signs to get a decent model. We feel like we can also kind of recover which points give what classification from this, which makes it more prone to false positives. In this case, we have that false negatives are worse than false positives, because if the particle is alive and we classified it as dead, we just missed a pretty important piece of information.

We learned that trees and gridsearch have the option to be parallelized with sklearn and xgboost and such. However, everything was already working fast enough, so we did not consider updating this.

Visualizing and plotting the coordinate data was essential towards engineering features. One distinct characteristic of motile particles was their non-linearity. This feature that we came up with was the linearity of the data, and we have the correlation coefficient to see if the particle was traveling in a true straight line. This ended up being a pretty important feature. One feature that we wanted to implement was to see the how predictable the following points would be by a RNN, but we didn't really have time for that. We thought this would improve our model because if the RNN could predict where the position would be in some time, the particle is predictable and is less likely to be a living being.

What we found interesting was how much XGBoost weighted the importance of the mean step speed. We wish we knew why, but we think that it has something to do with the infinite amount of ways to split a decision tree. We were thinking that it's because the mean step speed is the first item, and when we removed it, the first item was still important. Removing the first two columns gave us a more regular feature importance list, but I'm not sure. We were thinking of changing the impurity measure, to maybe possibly change the way XGBoost changes its splits, but we couldn't find the documentation to change the impurity measure.