Reconstructing Secrets: Using Machine Learning Models to Approximate Proprietary Whoop™ Algorithms

Tyler Anderton

Abstract

Wearable fitness devices, such as the Whoop device, are now widely-used products for monitoring health and optimizing athletic performance. This project focuses on leveraging personal Whoop data collected over 21 months to achieve two objectives. First, I aim to accurately predict the labels of various activities recorded within the Whoop app. Second, I seek to approximate Whoop's proprietary Recovery scores and obtain insights into the key features and relative complexity of Whoop's algorithm.

The dataset for this project was generated from an export of my personal data from the Whoop app and contains metrics for logged activities, daily aggregate physiological data, and self-reported behaviors. The activity classification task was performed on the set of logged activities and involved the training of multiple classifiers on two uniquely-labeled subsets of the data. The recovery score regression task utilized the set of daily aggregate physiological data and included feature selection to identify the metrics essential to Whoop's proprietary algorithm.

The results demonstrate that the simple neural network outperformed traditional models in activity classification, especially on the subset with a larger variety of class labels. In the regression task, the simple neural network underperformed, while the Lasso and Ridge models performed the best, both out of the box and after feature selection. This indicates the potentially simple nature of Whoop's Recovery score algorithm, and highlights the small subset of this algorithm's likely critical features.

1 Introduction

1.1 Background and Scope

Wearable fitness trackers have become increasingly popular for monitoring health and performance metrics (Piwek et al. 2016). The Whoop device, in particular, is a leader in the wearable fitness device space, marketing primarily to athletes to aid in optimizing their recovery and performance (Roberson 2021). The current version of the Whoop device, the Whoop 4.0, uses optical sensors to track heart rate, skin temperature, and blood oxygen saturation (Whoop). These raw data then sync to the Whoop mobile-device app, where they are sent to Whoop's servers and used to calculate cardiovascular fitness metrics like heart rate variability (HRV) (Shaffer & Ginsberg 2017) and resting heart rate (RHR). These data are also fed into proprietary algorithms to estimate various health measures like respiratory rate, calories burned, and even to identify periods of physical activity, the amount of sleep needed, and the time spent in each stage of sleep each night. Finally, these data are used to generate Whoop's trademark Strain and Recovery scores, which give the user

objective measurements of their athletic performance and readiness (Whoop).

The aim of this project is two-fold. First it aims to demonstrate the ability to accurately predict the labels of activities that were recorded and manually labeled within the Whoop app. This part of this project was in part inspired by a 2020 study that utilized wearable fitness tracker data to predict physical activity types (Fuller et al. 2020), and is hereon referenced as “Activity Classification” for clarity. The second objective is to use various machine learning models to approximate Whoop’s proprietary algorithms for their Strain and Recovery scores, thereby identifying some intuition for the key features used in these algorithms and their relative complexity with respect to these features. This part of the project has been named “Recovery Score Regression”.

1.2 Whoop Dataset

This project was completed using my personal Whoop data, collected over the span of about 21 months between September 2022 and June 2024. These data were then exported from the Whoop app, which provides three .csv files that contain distinct aggregate datasets of interest.

The first data file is called `workouts.csv` and contains aggregate metrics for each activity logged in the Whoop app. These metrics include features like the workout start time and duration, calories burned, average and max heart rates, and the time spent in each heart rate zone (Edwards 1993). Each logged activity is also labeled with the activity name and Whoop’s Strain score for that activity.

The second data file is called `physiological_cycles.csv` and contains daily aggregate metrics that can be split into two subcategories. First, Strain metrics include aggregate values like total calories burned, total daily Strain, and average and max heart rates for the entire day. These features can also be joined and compared with the metrics collected for each individual activity on any given day. Second, Recovery metrics are collected during the following night’s sleep, and include features like RHR, HRV, skin temperature, respiratory rate, sleep onset time, total duration in bed, and the duration of that time spent awake and in each stage of sleep (Dement & Kleitman 1957). This Recovery subcategory also includes Whoop’s proprietary Recovery score that this project attempts to estimate as a regression target.

The third and final data file is called `journal_entries.csv` and contains daily boolean answers to customizable questions about behaviors like 'Have any alcoholic drinks?', 'Have any caffeine?', and 'Avoid consuming processed foods?'. Hypothetically, the answers to some of these questions may have significant impact on Recovery Scores, and initially I was very interested to include them as features in my regression models. The app also allows the user to track specific values for some of these questions, like the quantity of alcoholic drinks or servings of caffeine that were consumed and at what times, but unfortunately, only boolean True/False answers were included in the data export from Whoop. Furthermore, none of these questions were included for all of the days studied in this project, leaving many null entries that either broke or negatively impacted the performance of my preliminary regression models. Given this problem with null answers and the lack of specific values that would have been most impactful on my models, I ultimately decided to exclude all of these journal entries from my feature set for this project.

2 Methods

2.1 Activity Classification

2.1.1 Preprocessing

First I dropped irrelevant features from the workouts dataset like the measured altitude change (which was always 0) and various features on which I didn't want my models to depend, like datetime features (datetimes were initially encoded, but later dropped before model training), activity duration (as the model should ideally identify an activity regardless of its duration), and Whoop's proprietary Strain score. I was left with the following metrics for each activity: calories burned, max HR, avg HR, and the percent of time spent in each HR zone 1-5. Perhaps further feature engineering and selection could have been helpful to fully optimize performance, but I felt satisfied with these features for the scope of this project (Kuhn & Johnson 2019).

Since these data are from a single subject (me) who has obviously tended toward a few activities much more than others, I needed to address the issue of class balance (Krawczyk 2016). The full workout dataset included 18 unique features with counts ranging from 1 to 161. I took two separate approaches to managing this class imbalance and compared the results between the two.

For the first approach, I kept all data points, but I combined all classes into 3 similar artificial labels, hence this approach was named "Full 3". "Powerlifting" activities were relabeled to "Weightlifting", "Jiu Jitsu", ""Wrestling", "Kickboxing", and "Boxing" were relabeled to "Martial Arts", and finally all other activities, including a wide variety ranging from "Yard Work" and "Yoga" to "Box Fitness", were renamed to a generic "Other" label. In the end, the "Weightlifting" class included 196 entries, the ""Martial Arts" class included 144 entries, and "Other" included 109 entries, still not achieving perfect class balance, but one much better than in the original dataset.

The first approach has obvious downsides of course, most obvious being that a prediction between just two real classes and a third "Other" class is not very applicable to a product meant for a general population that partakes in many other activities. In an attempt to demonstrate classification on a broader set of classes, for my second approach I simply truncated my class set by removing any classes with a count less than 10. This left me with a still-unbalanced set of 8 distinct classes, resulting in the approach name "Limited 8". This set still included class counts ranging between 12 and 161, but at least this dataset would eliminate the worst outliers and ensure compatibility with the cross validation methods implemented later during model training.

Finally, for each approach I encoded the class labels with `sklearn LabelEncoder` for model training.

2.1.2 Model Selection

I chose four classification models to train and compare on this task. Firstly I chose to implement a custom `RotationForest` classifier (Rodriguez et al. 2006), since Fuller et al. 2020 achieved their best performance using this architecture in their own activity classification task. I also included a `RandomForestClassifier` to provide a baseline to the Rotation Forest. XGBoost models are widely popular for their classification performance (Chen & Guestrin 2016), so I decided to test an `XGBClassifier` for this task as well. All of these models were implemented using the `sklearn` and `xgboost` python packages.

As a comparison to these standard models (hereon dubbed "ML Models" for brevity), I also implemented and trained a simple

PyTorch fully-connected neural network with two hidden Linear layers, a single BatchNorm1d layer, and ReLU activation layers. The point of this model was not to test the limits and capabilities of state-of-the-art neural model architecture design, but simply compare the performance of a simple network to that of well-established Forest and XGBoost models.

2.1.3 Model Training

For all models, sklearn StandardScaler was used to standardize the features. For the ML Models, I used StratifiedKFold with n_splits=10 and cross_val_predict and cross_val_score to evaluate performance based on the mean overall accuracy. For the neural model, to be completely comparable, I could have implemented a version of these cross-validation methods, but for the sake of simplicity I instead used train_test_split with stratification and test_size=0.2. Here I also used the class weights with CrossEntropyLoss and the standard Adam optimizer, again evaluating performance on the mean accuracy. Neural models were trained for a maximum of 200 epochs, but with early stopping after 10 epochs of no improvement. Performance metrics for all models were logged and compared with Tensorboard.

For the first round of training, all models were left with their default hyperparameters, shown in **Figure 3**, to obtain baseline performance metrics. After these baseline accuracy scores were observed, Grid Search was then implemented on each model to optimize performance on each dataset, with unique hyperparameters found for each model for each of the Full 3 and Limited 8 datasets. GridSearchCV was used to tune the ML models, and ray.tune was used to optimize the neural model.

2.2 Recovery Score Regression

2.2.1 Preprocessing

As with the classification task, here I also started by dropping any irrelevant or clearly redundant features from the physiology dataset. This initially included fields like the timezone and cycle end time/wake time, and would later also include most datetime metrics and Whoop's proprietary Strain score. Current and previous night sleep onset times were kept and encoded on a 24 hour cycle.

After dropping the immediately unnecessary features, all percent values were standardized to the range [0,1], and the fields representing the duration in minutes of different stages of sleep were converted to the percent of the total time in bed. These percentages seemed a more obvious way to represent the quality of each night's sleep, and could recover the raw duration information when combined within the model with the total duration spent in bed.

As previously discussed, the features within the physiological dataset can be split into two subcategories: Recovery and Strain metrics. As exported from Whoop, each datapoint includes Recovery metrics for the previous night and Strain metrics for the following day, ostensibly to align with Whoop's stated goals to assist athletes in optimizing their performance given the previous night's recovery (Whoop). However, for the purposes of this project, the Strain metrics for the preceding day needed to instead be aligned to the corresponding Recovery metrics for the following night. This required splitting the physiological dataset into each of its two subcategorical sets, adjusting the dates within the recovery dataset back one calendar day, and then rejoining the datasets on the adjusted date values.

Similar to the Activity Classification task, the Recovery Score Regression was performed by two slightly differing methods in parallel, this time differentiated by unique feature sets rather than different labels. The first method used only the Strain and Recovery metrics included in the physiology dataset to predict the Recovery score. After preparing the physiological (PHYS) dataset for this first method, the workout dataset was merged to provide additional information for the models and to test for improvements in performance. Since each day could have 0, 1, or more logged activities, the workout information needed to be aggregated daily before joining to the physiological dataset. First the percentages of time spent in each heart rate zone were converted to total durations in minutes. Then the total workout duration, calories burned, and the times spent in each heart rate zone were summed together, and the maximum and average activity heart rates were recorded. Initially the activity Strain scores were also aggregated, but these proprietary metrics were dropped before training the models. The activity name encodings were aggregated as lists, and these lists were then encoded into unique integers with `LabelEncoder`. Finally all of these aggregated features were joined by the date to the physiological dataset to create the MERGE dataset.

2.2.2 Model Selection

Five regression model architectures were trained and evaluated on this task. Similar to the classification task, four classic "ML models" were compared to the performance of a simple neural net. Firstly, `Lasso` and `Ridge` were selected as simple baselines for any regression task (Hastie et al. 2009). Then `RandomForestRegressor` and `XGBRegressor` were chosen for their proven efficiency and efficacy in regression tasks like these (Chen & Guestrin 2016). Finally, the regression neural net shared exactly the same architecture as the classification model, except with only a single output dimension. As with the classification task, testing advanced or differing neural architectures was beyond the scope of this project.

2.2.3 Model Training

Again, I used `sklearn StandardScaler` to normalize the features for all models. Here too, cross validation was implemented for the ML models with `KFold`, `n_splits=10`, and `cross_val_predict` to record `y_pred` prediction values. `y_pred` was then used to record MSE, MAE, and R^2 metrics, with MSE being the regression optimization metric. For the neural model, I again used `train_test_split` with `test_size=0.2` and the `Adam` optimizer, but this time with `MSELoss` as the loss function. Neural models were again trained for a maximum of 200 epochs with early stopping with patience of 10 epochs. Performance metrics for all models were again logged and compared with `Tensorboard`.

As with the classification task, all models were first trained with their default hyperparameters, shown in **Figure 6**, to obtain a baseline of performance. After these baseline errors were observed, Grid Search was then implemented on each model to optimize performance on each dataset, with unique hyperparameters found for each of the feature sets: PHYS and MERGE. Here too I used `GridSearchCV` to optimize the ML models and `ray.tune` for the neural model. Finally, feature importances were collected from the best performing ML models and averaged to rank the most important features across these models. Only the best half of the features (for each of the PHYS and MERGE sets) were selected and used to perform a final

round of hyperparameter tuning and evaluation for all five models.

3 Results

3.1 Activity Classification

Figure 1 contains the baseline and best model accuracies measured for each model on each dataset, along with the improvements in accuracy observed. These accuracy values are also visualized in **Figure 2**. On both datasets, the SimpleNN model performed the best both before and after hyperparameter tuning. The neural net also improved the most from the tuning process on the Limited 8 dataset, while the XGBoost Classifier benefited the most on the Full 3 dataset.

Figure 3 shows a comparison between the default and optimized hyperparameters for each classification model. It is possible that the reduction in `n_estimators` from 100 down to 50 for the Full 3 Random Forest model drove the odd reduction in performance from 0.7619 to 0.7396 accuracy, but it is hard to tell for sure. What is more clear is that XGBoost benefited from

dataset	model	base_acc	tuned_acc	improvement
Full 3	Random Forest	0.7619	0.7396	-0.0223
	Rotation Forest	0.7441	0.773	0.0289
	XGBoost	0.7486	0.7886	0.04
	SimpleNN	0.7796	0.7896	0.01
Limited 8	Random Forest	0.6711	0.6787	0.0076
	Rotation Forest	0.6605	0.6945	0.034
	XGBoost	0.6479	0.6971	0.0492
	SimpleNN	0.7232	0.7924	0.0692

Figure 1. Mean accuracies achieved for each classification model. “Full 3” and “Limited 8” refer to the distinct class label sets, while “base_acc” and “tune_acc” refer to the models’ performance before and after hyperparameter tuning.

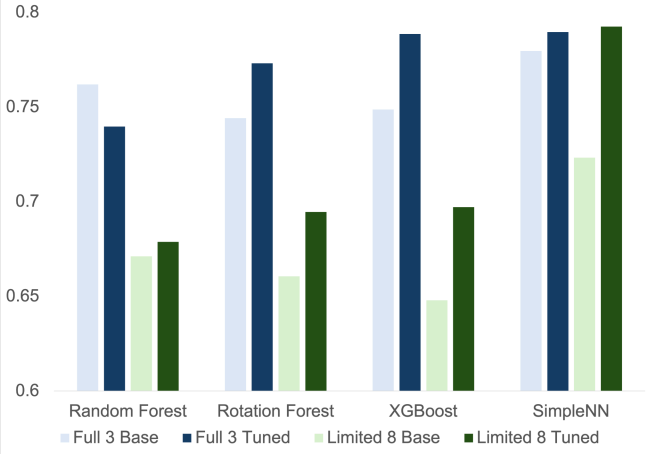

Figure 2. Mean accuracies achieved for each classification model. “Full 3” and “Limited 8” refer to the distinct class label sets, while “Base” and “Tuned” refer to the models’ performance before and after hyperparameter tuning.

		Default	Full 3	Limited 8
Random Forest	n_estimators	100	50	200
	max_depth	None	40	20
	min_samples_split	2	2	2
	min_samples_leaf	1	1	2
Rotation Forest	n_classifiers	10	20	20
	n_splits	3	7	7
	base_classifier__n_estimators	100	50	100
	base_classifier__max_depth	None	None	20
	base_classifier__min_samples_split	2	5	2
XGBoost	n_estimators	100	100	100
	learning_rate	3e-1	5e-3	1e-1
	max_depth	6	11	7
	subsample	1.0	0.9	1.0
	colsample_bytree	1.0	0.9	1.0
	gamma	0	0	0
SimpleNN	learning_rate	1e-2	1e-3	1e-3
	hidden_size	1000	100	1000
	batch_size	32	16	64

Figure 3. Comparison between the default and optimal hyperparameters found for each model, for each of the Full 3 and Limited 8 class sets.

increasing its `max_depth` on both datasets. Additionally while it appears the `hidden_size` parameter had little effect on the accuracy of the SimpleNN model, the tuner found greater success by adjusting the `learning_rate` and `batch_size` parameters for the Limited 8 dataset, likely negating some overfitting that occurred with the default parameters.

3.2 Recovery Score Regression

Figure 4 displays the baseline, tuned, and "select" (with feature selection) MSE values recorded for each model, trained on each dataset, as well as the improvements observed with each iteration. These MSE values can also be compared graphically in **Figure 5**. Interestingly, the SimpleNN severely underperformed compared to the ML Models, even after hyperparameter tuning and feature selection. It is difficult to understand why without further investigation, but one could speculate that without decreasing the learning rate, the neural model simply could never converge close enough to the optimal MSE values to compete with the ML models. Although the neural model did benefit the most from tuning, this affected only a very moderate change in MSE, and feature selection only worsened this effect. The only other model to improve significantly (although still very slightly) was XGBoost, which went from the worst of the ML Models to the best after hyperparameter tuning.

Figure 6 shows a comparison between the default and optimized hyperparameters found after the tuning and feature selection steps for each regression model. The SimpleNN was the only model to see somewhat significant changes in MSE from hyperparameter tuning, with most of this improvement again coming from changes in the `learning_rate` and `batch_size` parameters to reduce overfitting. It seems that when the feature sets were reduced, the tuner had a difficult time managing the loss in information, decreasing the `hidden_size` for the PHYS dataset, increasing the `hidden_size` for the MERGE dataset, and yielding increased MSE in both cases.

Even though the SimpleNN model performed slightly worse after removing the least important half of features, none of the ML models yielded the same drop in performance, all producing ever so slightly decreased MSE values. This suggests that the dropped features indeed were not important for predicting Recovery scores, and that the remaining features (or even smaller subsets thereof) are sufficient for estimating Whoop's proprietary algorithm.

dataset	model	base_mse	tuned_mse	tuned_imprvmt	select_mse	select_imprvmt
PHYS	Lasso	0.0101	0.0101	0	0.0099	-0.0002
	Ridge	0.0103	0.0102	-0.0001	0.0099	-0.0004
	Random Forest	0.0101	0.0101	0	0.0098	-0.0003
	XGBoost	0.0113	0.0092	-0.0021	0.009	-0.0023
	SimpleNN	0.0523	0.0338	-0.0185	0.0363	-0.016
MERGE	Lasso	0.0101	0.0101	0	0.0099	-0.0002
	Ridge	0.0102	0.0105	0.0003	0.01	-0.0002
	Random Forest	0.0102	0.01	-0.0002	0.01	-0.0002
	XGBoost	0.0114	0.0091	-0.0023	0.0088	-0.0026
	SimpleNN	0.0513	0.0338	-0.0175	0.0405	-0.0108

Figure 4. Best MSE values achieved for each regression model. "PHYS" and "MERGE" refer to the distinct feature sets, while "base_mse" refers to the models' performance before the first round of hyperparameter tuning, and "tuned_mse" refers to the performance after the first round of tuning, and "select_mse" refers to the final values after performing feature selection and the final round of hyperparameter tuning. Each "imprvmnt" column displays the difference in MSE values between the baseline and their respective round of tuning.

The features for each dataset are shown with their relative average importance ranks in **Figure 7**, with the top half of features (the ones kept in the final models) highlighted in ***bold***. We can see a large overlap between the selected feature sets between PHYS and MERGE, with all features from the selected PHYS dataset also being selected for the MERGE models, only excluding `in_bed_min` by a single position.

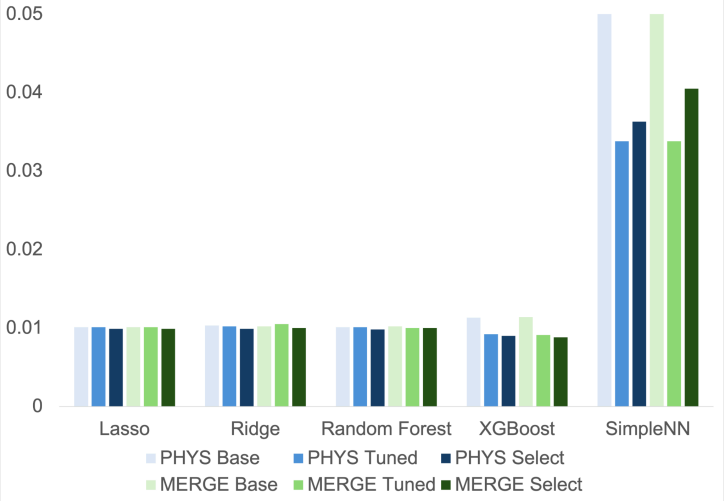

Figure 5. Best MSE values achieved for each regression model. "PHYS" and "MERGE" refer to the distinct feature sets, while "Base" refers to the models' performance before the first round of hyperparameter tuning, and "Tuned" refers to the performance after the first round of tuning, and "Select" refers to the final values after performing feature selection and the final round of hyperparameter tuning.

		Default	PHYS Tuned	PHYS Select	MERGE Tuned	MERGE Select
Lasso	alpha	1	1e-3	1e-4	1e-3	1e-4
	max_iter	1000	100	500	100	500
	tol	1e-4	1e-6	1e-3	1e-6	1e-5
Ridge	alpha	1	1e-3	1e-3	1	1e-1
	max_iter	None	1000	1000	500	1000
	tol	1e-4	5e-2	5e-2	5e-2	5e-2
	solver	auto	saga	saga	sag	sag
Random Forest	n_estimators	100	50	40	40	30
	max_depth	None	15	10	20	None
	min_samples_split	2	3	2	3	2
	min_samples_leaf	1	2	1	2	2
XGBoost	n_estimators	100	100	125	150	150
	learning_rate	3e-1	1e-1	1e-1	5e-2	1e-1
	max_depth	6	3	3	4	2
	subsample	1.0	0.8	0.8	0.6	0.8
	colsample_bytree	1.0	0.8	0.8	1.0	0.8
	gamma	0	0	0	0	0
SimpleNN	learning_rate	1e-2	1e-1	1e-1	1e-1	1e-1
	hidden_size	1000	1000	500	1000	5000
	batch_size	32	64	64	32	64

Figure 6. Comparison between the default and optimal hyperparameters found for each model at each stage of tuning, for each of the PHYS and MERGE feature sets.

PHYS		MERGE	
feature	avg_rank	feature	avg_rank
hr_var	***0***	***hr_var***	***0.25***
calories	***1.5***	***resp_rate***	***4.25***
resp_rate	***3***	***calories***	***4.75***
hr_rest	***4.25***	***sleep_onset_min_sin***	***5.75***
sleep_onset_min_sin	***4.25***	***hr_rest***	***9***
sleep_consistency_p	***8***	***activity_hr_max***	***10.75***
in_bed_min	***9.25***	***sleep_consistency_p***	***12.5***
hr_avg	***9.5***	***hr_avg***	***13***
skin_temp_c	***9.75***	***activity_codes_comb***	***13.25***
awake_p	***10***	***skin_temp_c***	***13.75***
restorative_sleep_p	***10.25***	***activity_duration_min***	***13.75***
hr_max	***11***	***activity_hr_zone_3_min***	***14***
rem_sleep_p	11.25	***awake_p***	***14***
sleep_efficiency_p	12.25	***activity_hr_zone_2_min***	***14.25***
deep_sleep_p	13.5	***restorative_sleep_p***	***14.5***
sleep_performance_p	13.75	***hr_max***	***15***
light_sleep_p	15.75	***rem_sleep_p***	***15.5***
prev_sleep_onset_min_sin	16.25	in_bed_min	17.25
blood_ox_p	16.25	activity_calories	18
prev_sleep_onset_min_cos	16.5	activity_hr_zone_1_min	19.25
sleep_need_min	17.5	blood_ox_p	19.5
sleep_debt_min	18.5	prev_sleep_onset_min_cos	20
sleep_onset_min_cos	20.75	light_sleep_p	20.5
		prev_sleep_onset_min_sin	21.5
		sleep_performance_p	21.75
		deep_sleep_p	21.75
		sleep_efficiency_p	21.75
		activity_hr_zone_4_min	22.25
		sleep_need_min	22.25
		sleep_onset_min_cos	23.75
		activity_hr_avg	24
		sleep_debt_min	24.25
		activity_hr_zone_5_min	26.25
		activity_count	28.75

Figure 7. Comparison of the average feature importance rankings between the PHYS and MERGE datasets. Only the top half of features for each dataset (in ***bold***) were kept for the final “Select” tuning stage.

4 Discussion

This project was successful in achieving its stated goals to demonstrate efficacy in activity classification and in the estimation of Whoop’s proprietary Recovery score, even while being heavily limited by the narrow scope of data available.

On the activity classification task, the neural and XGBoost models both performed well when the class labels were aggregated into 3 artificial labels. XGBoost didn’t perform as well on the limited dataset with 8 classes, but the neural model still outperformed all the others. On both datasets the SimpleNN significantly outperformed the Random and Rotation Forests favored by Fuller et al. This indicates that for applications with a sufficient computation budget, small neural models may outperform the traditional Forest and XGBoost classification models.

While the SimpleNN did not perform well on the Recovery score regression task, this project was still successful in approximating Whoop’s proprietary algorithm, achieving MSE values on the order of 1e-2 on a variety of machine learning models. Notably, Lasso and Ridge performed basically as well as the more powerful XGBoost, with a difference in MSE of only roughly 1e-3. This indicates that Whoop’s algorithm is likely a relatively simple function of the features explored in this project.

As mentioned, these successes do not come without caveats. Having a subject sample size of just 1 and with data only spanning less than 2 years makes for an interesting case study, but more work is required to confirm the findings here generalize to a wider population.

Firstly, Whoop’s entire customer base (and source of data on which they design their algorithms) undoubtedly has much more variation in their measurement values and the ranges of activities logged. As discussed, with my personal data alone I was only able to train models to classify either 3 or 8 activity labels; however, Whoop has an ever-increasing number of possible activity labels in their app, ranging from “Cuddling with Child” and “Stroller Walking” through “Disk Golf” and “Horseback Riding” to “Triathlon” and “Track and Field” and dozens more. Ostensibly, they achieve relatively good classification accuracy on each of these labels. Perhaps they leverage additional fine-tuning based on each user’s data to achieve this, but even so, models trained only on my personal activities could never hope to generalize so well.

Secondly, Whoop also has access to minute-by-minute measurements of heart

rate, oxygen saturation, respiratory rate, etc that surely allow for much finer modeling. This project was instead limited to the aggregate data available through the Whoop app export, which at its most detailed level, still only gives aggregate metrics for individual activities. These data give no information about how any measurements may change minute-to-minute, which surely would help a model determine activity type and overall Strain. These time-based measurements could also affect the Recovery score, as the variabilities, not just the averages, of many of these measurements are hypothetically impactful on a user's recovery.

I would love to repeat this project with access to a larger, more complete set of users' data. Such a project would promise better generalization to a wider customer base and could also allow for the implementation of more advanced modeling techniques, such as Apple's Hybrid Modeling approach (Nazaret et al. 2024). A more detailed feature set could also allow for better feature engineering to include metrics like the momentary Karvonen Intensity (Karvonen et al. 1957) or variability metrics like activity HRV or respiratory rate variability. Such features could allow better modeling of activity features for identification, as well as Strain and Recovery scores that could possibly even improve on Whoop's current algorithms.

References

Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794. https://doi.org/10.1145/2939672.2939785

Dement, W., & Kleitman, N. (1957). *Cyclic variations in EEG during sleep and their relation to eye movements, body motility, and dreaming*. Electroencephalography and Clinical Neurophysiology, 9(4), 673-690. https://doi.org/10.1016/0013-4694(57)90088-3

Edwards, S. (1993). *The Heart Rate Monitor Book*. Sacramento, CA: Fleet Feet Press.

Fuller, D., Cummins, C., Matwijiw, T., & Leonard, J. (2020). *Using machine learning methods to predict physical activity types with Apple Watch and Fitbit data using indirect calorimetry as the criterion*. Research Square. https://doi.org/10.21203/rs.3.rs-17022/v1

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.)*. Springer Science & Business Media.

Karvonen, M. J., Kentala, E., & Mustala, O. (1957). *The effects of training on heart rate; a longitudinal study*. Annals of Medicine and Experimental Biology of Finland, 35(3), 307-315.

Krawczyk, B. (2016). *Learning from imbalanced data: open challenges and future directions*. Progress in Artificial Intelligence, 5(4), 221-232. https://doi.org/10.1007/s13748-016-0094-0

Kuhn, M., & Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press.

Nazaret, R., Scott, M., & Zhao, Z. (2024). *Personalized Heart Rate Estimation Using Hybrid Neural Networks*. Apple Machine Learning Research.

https://machinelearning.apple.com/research/personalized-heartrate

Piwek, L., Ellis, D. A., Andrews, S., & Joinson, A. (2016). *The Rise of Consumer Health Wearables: Promises and Barriers. PLOS Medicine*, 13(2), e1001953. https://doi.org/10.1371/journal.pmed.1001953

Roberson, M. (2021, May 24). *Whoop Becomes Official Fitness Wearable Of PGA Tour*. Forbes. https://www.forbes.com/sites/michaelroberson/2021/05/24/whoop-becomes-official-fitness-wearable-of-pga-tour

Rodriguez, J. J., Kuncheva, L. I., & Alonso, C. J. (2006). *Rotation Forest: A New Classifier Ensemble Method*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(10), 1619-1630. https://doi.org/10.1109/TPAMI.2006.211

Shaffer, F., & Ginsberg, J. P. (2017). *An Overview of Heart Rate Variability Metrics and Norms. Frontiers in Public Health*, 5, 258. https://doi.org/10.3389/fpubh.2017.00258

Whoop. (n.d.). *WHOOP 4.0: The World's Most Powerful Fitness and Health Wearable*. https://www.whoop.com/