# Smartphone User Identification

*Classify smartphone users using accelerometer information*

After trying a multitude of models and features in a total of 32 submissions, I placed #11 on the private leaderboard, with a score of 0.93325. I will be detailing the models I tried out, what worked, what didn't, what sort of features I used, and my thoughts and reasoning during the process.

## Submitted models

My two best models are both soft voting models, but the estimators used could not be more different – one is a 4-way voting ensemble made of a Logistic Regression (LR) model, a Support Vector Machine (SVM) model, a Random Forest (RF) model, and a model of Gradient-Boosted trees, more specifically using the LightGBM (LGBM) framework; the other model is a combination of two Neural Networks – one is a simple NN with merely one layer, the other one is a complex 1D convolutional neural network (CNN).

The first model came to be as an effort of previous used models which gave relatively good accuracy, particularly LR and SVM which generalized well to the testing dataset. For LR, SVM and RF I used sklearn's GridSearchCV to optimize hyperparameters, with a 5-fold cross-validation, while for LGBM I opted for using Optuna, a library designed for fast hyperparameter optimization. All these models are trained on the same set of features, which are modelled statistically, which will be further detailed later.

In the second model both networks extensively use batch normalization, Adam as optimizer, and early stopping on lack of loss improvement for 10 epochs, first being trained on the statistic features, while the convolutional network uses raw data. Both models have had batch sizes, learning rates and dropouts tested by trial and error, but also the architecture and layer parameters. Both models use softmax as ouput on 20 layers, each corresponding to one of the 20 classes.

Most of the models I used in the beginning were the ones that ended up going in the first voting classifier, with different features, as my feature set grew over time, or with different parameters before I optimized them. I included one of my very first models, a 5-NN on only the average of the raw data, and a SVM when I was trying to understand the fourier transform, with like half of the final features, representing my progress at various points of the competition.

## What didn't work

Model performance directly depends on the set of features used, and basically all models performed worse on the raw data as opposed to statistic features. On the statistic features, particularly disappointing were k-NN for which I tried with all sorts of k, and the gradient boosted trees, which seem to overfit pretty hard on the training data. Similarly Naïve-Bayes gave pretty poor results, and so did Ridge Regression classifier.

Some models also failed to work due to parameters – I spent almost the entire competition thinking that neural nets don't do well because I was using SGD as opposed to Adam, and because I had no batch normalization.

Also using scikit's StandardScaler() did not work at all, dropping the validation accuracy of 95% models into 40% when applied to the raw data.

## Features

For both sets of features, the raw data but also the statistics, I needed to make sure that the input size dimension is fixed as to not run into size compatibility problems, so I considered only the first 130 rows. 130 because I didn't check for the minimum row size of all files and it seemed like a good enough approximation.

For the statistic set, I used mean, std, min, max, percentiles at 25%, 50% and 75%, argmax, skew, kurtosis, and the difference between max and min, directly on each axis of the raw data. I also made use of the number of flips the graph makes (scaled as to weigh less for the SVM), the first two highest frequencies (also scaled) via Fourier Fast Transform (FFT) for each axis and their amplitudes. Another significant part of the feature set was making use of the absolute difference $|row(i+1) - row(i)|$, and taking mean, std, max and percentiles on it too – no minimum as it tended to be 0. Similarly, I took mean, std, min, max on the L2 distance of each coordinate set, so $\sqrt{x^2 + y^2 + z^2}$, and added those as features.

As those were all global features, up next I added local features too – I took a window of 26 points (corresponding to 26 ms, roughly a quarter of a second) and obtained features on this data alone – the mean of the raw data, the mean of distance data, and the mean of the L2 distance. This set of features could've been expanded further (percentiles, max – min, std), but it was reporting worse on validation and I ran out of submissions, so I settled on mean alone.

So I had 61 global features, 55 statistic in nature and 6 from FFT, and 35 from local features, for a total of 96 features.

## Hyperparameters

Both voting models use soft voting, i.e. probabilities – tried hard voting but it was worse in terms of performance. As I mentioned previously, first model used GridSearchCV on the entire training set for hyperparameter tuning, except for the LGBM, for which I chose Optuna instead.

For Logistic Regression I settled on using the newton-cg solver for highest computation fidelity, even though it's substantially slower. Accuracy still increases with number of max iterations during grid search, so I settled on 3000 iterations (15 min). High C gives best results, settled on C=1000.

For SVM, similar to LR, higher C is better, so C=1000. For kernel choices, linear is better than rbf at low C values, but when C is high, they give very similar results – settled on using rbf. Also tried LinearSVC but results were way worse. I set probability=True to internally use CV.

For RF what's best isn't exactly clear due to seeding, but 180-200 trees and sqrt max features seem best. Also, for criterion both log_loss and gini performed best.

For LGBM, I varied number of leaves, regularization parameters, number of minimum child samples, feature/bagging fractions, and bagging frequency. I settled on the hyperparameters which returned highest validation accuracy. The parameters are long digit-wise and rather unwieldy/non-descriptive, so I left them out of the documentation.

For the neural networks I have no explanation for hyperparameter tuning besides trial and error – add more dropout, add more dense layers, remove dense layers, add more dropout layers, lower dropout rate, lower learning rate, increase batch size, etc. Batch normalization made a large difference, so did adding more dropout layers, and also lowering the learning rate. Settled for 64 batch size, learning rate $\approx 10^{-4}$, dropout values of 20%/30%, and 150 epochs (usually stops at around epoch 80-120).

The architecture for the CNN was inspired by the paper listed in *Resources* chapter, on which I changed filter and kernel sizes, and also played with activation and padding types until I found best validation accuracy. Both neural nets use early stopping on validation loss when preparing for validation, and on the training loss when preparing for testing, stopping for patience=10.

## Results

All models are ran with optimal hyperparameters on 5-CV, except for LR on Voting for which I used 1000 max_iter as opposed to 3000, due to a lack of patience. Similarly for LR and Voting on validation accuracy.

| | LR | SVM | RF | LGBM | Voting |
|---|---|---|---|---|---|
| 5-CV acc | 96.27 | 96.56 | 97.04 | 97.71 | 98.03 |
| Valid acc | 95.40 | 96.30 | 96.75 | 97.75 | 97.40 |

It can be seen that cross-validation accuracy isn't that much different from validation accuracy, and also that voting accuracy tends to be largest on 5-CV. On validation it might appear lower but LGBM overfits on the training set, while the voting model generalizes better to the test dataset – 91.47 vs 85.85 on private leaderboard.

I tried cross validation for my neural nets too, but something is going wrong about accuracy, either the type or the way it's calculated, or the splits, or overfitting, I don't know, but I get 100 accuracy on some of them and that's rather worrying, so I'm only including validation acc.

| | CNN | NN | Voting |
|---|---|---|---|
| Valid acc | 98.05 | 97.40 | 98.30 |

The neural network based model performs better not only on validation, but also on testing – my highest submission is 93.32 using precisely this voting model, as opposed to the 91.47 on the other one. Also, after competition ended, I tested my networks with halved learning rate and achieved even better accuracy, 93.9 on private leaderboard 😢.

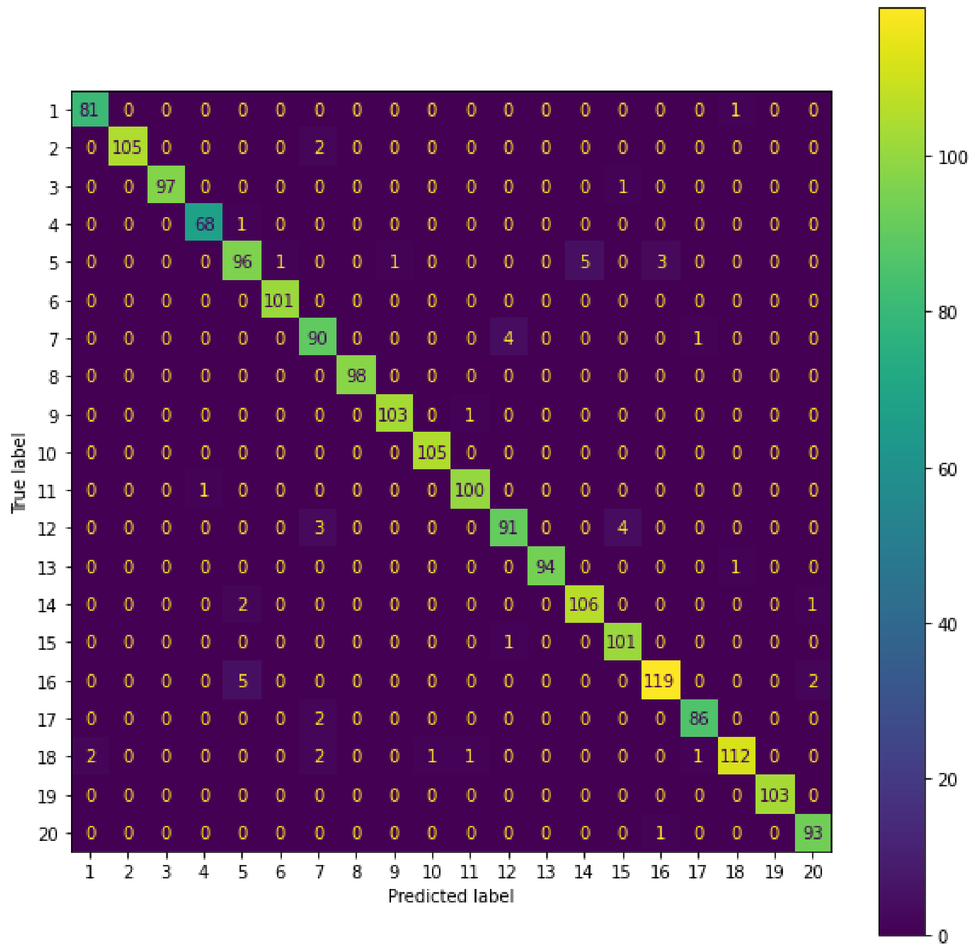
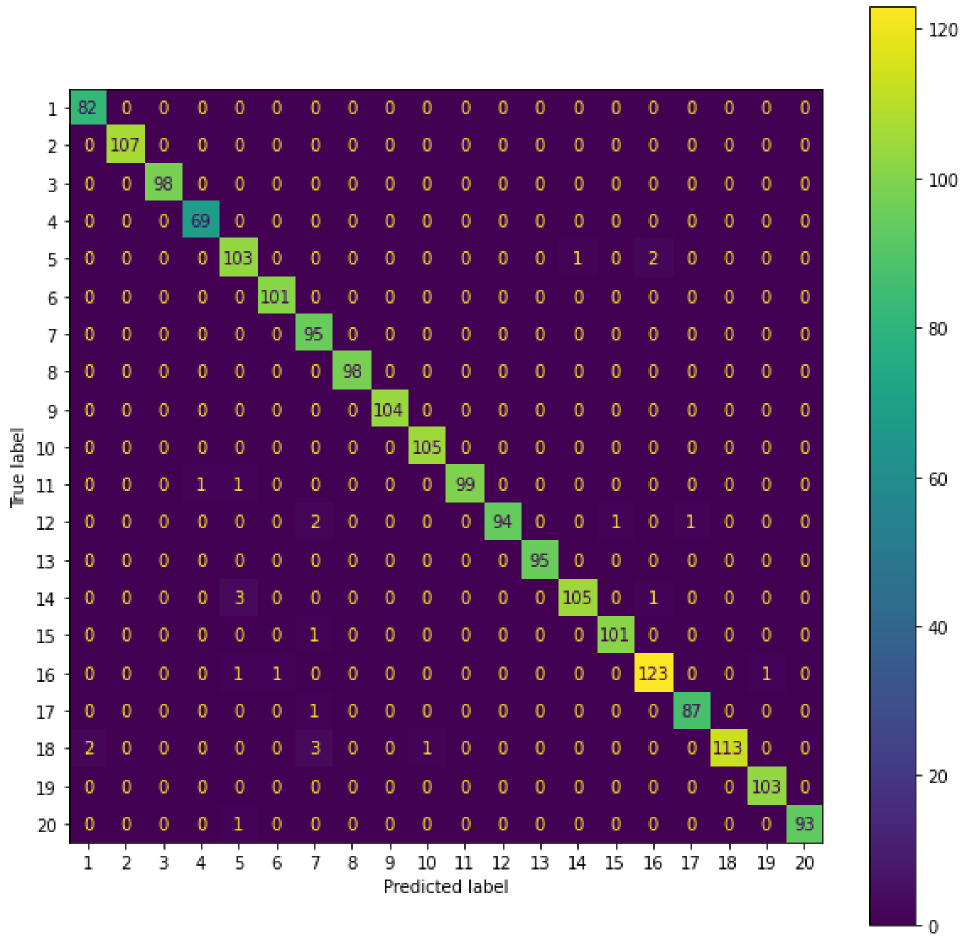


*Figure 1: First voting model*



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 82 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 107 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 69 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 103 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 101 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 95 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 104 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 105 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 99 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 94 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 95 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 105 | 0 | 1 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 101 | 0 | 0 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 123 | 0 | 0 | 1 | 0 |
| 17 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 87 | 0 | 0 | 0 |
| 18 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 113 | 0 | 0 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 103 | 0 |
| 20 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 93 |

*Figure 2: Second voting model*

It can be seen that second voting model struggles differentiating between classes 7 and 18, and 5 and 14. For first model there are more struggles, being overall the worse model. It can also be seen that there's a data imbalance for class 4 in the validation set, having way fewer samples than any other class.

## Resources

I extensively used scikit-learn for model selection and optimization, NumPy for statistic feature extraction, and TensorFlow for creating the neural nets. I also made use of SciPy for frequency-related features, LightGBM for the implementation they provide, and Optuna for optimizing said implementation. Also, for the NN and LGBM I used the official wrappers to integrate the models within scikit-learn's ecosystem.

I've done plenty of research on the subject of accelerometer-based classification, and these papers either substantially inspired my choices of features/models or were directly based off of them:

Fridriksdottir E. and Bonomi A., "Accelerometer-Based Human Activity Recognition for Patient Monitoring Using a Deep Neural Network", National Library of Medicine, *Sensors* journal, 10 November 2020, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7697281/

Kwapisz J., Weiss G. and Moore S., "Activity Recognition using Cell Phone Accelerometers", Fordham University, *SensorKDD* conference, 25 July 2010, https://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf

Tsokov S. et al, "Accelerometer-based human activity recognition using 1D convolutional neural network", IOP Conference, 2021, https://iopscience.iop.org/article/10.1088/1757-899X/1031/1/012062

Also, I used BEXGBoost's article on Optuna for choosing features and also learning how to use the framework - https://towardsdatascience.com/kagglers-guide-to-lightgbm-hyperparameter-tuning-with-optuna-in-2021-ed048d9838b5.