Activity Classification Using WISDM Smartphone And Smartwatch Activity And Biometrics Dataset

Shifa Shaikh

Anee Dudhia

San Jose State University

May 17, 2021

Abstract

This project is about classifying different physical activities recorded using a smart-phone and a smart-watch by the popular WISDM laboratories. The data-set contains time-series raw data recorded from an accelerometer and a gyroscope of both the phone and watch at the same time. We use only smart-watch data-set due to less computation availability to classify 18 different daily physical activities using 6 different types of classification models. We even perform feature extraction and elimination technique to reduce the over-fitting problem of the models. The main goal of the project is to study, analyze, implement and improve the classification models. We record around 56% highest accuracy using KNN classifier with $n = 25$ neighbors. We even conclude that the $Tree - based\ Models$ might perform better if we change some parameters during the training.

Keywords: WISDM, time-series, classification.

Activity Classification Using WISDM Smartphone And Smartwatch Activity And Biometrics Dataset

Contents

List of Figures

List of Tables

Data Set Description

We use the open data-set from UCI Machine Learning Repository, Weiss (2019). The data-set is collected from a smartwatch and a smartphone of 51 subjects doing 18 types of day-to-day activities. The activities include walking, sitting, eating pasta, clapping, etc. Each activity is recorded for 3 minutes for both the watch and the phone simultaneously. The raw data-set has 2 folders namely watch and phone. Each of these has 2 sub-folders: an accelerometer and gyroscope. The signals are recorded are converted into text files having subject ID,activity, timestamp, x, y, and z-axis. The inputs are the x,y, and z-axis from the accelerometer and gyroscope. With these inputs, we predict the activity by the subjects. The input data type is numerical data or precisely floats. The output is Activity code labeled using the English alphabets. We classify 18 types of activities. There are no non-null values or missing values. However, the number of data lines in each directory varies. The author, Weiss (2019) says that it is due to the imperfect data collection methods.

Data Visualization

In this section we visualize our given data. As noted earlier, there are 2 types of devices from which the signals are recorded. Due to less computation and processing power availability, we merely focus on one device. We will be focusing on the smart-watch data only from now. The data-set has 2 separate folders: Accelerometer sensor and Gyroscope sensor. .

We analyze the input data i.e. our X, Y and Z axes of both the sensors. The figure 1 on page 19 shows that the data has no outliers. Even figure 2 on page 20 shows the plots for gyroscope's axes has no outliers.

Further, let's analyze the output. The figures 3 and 4 on pages 20 and 21 respectively shows the counts of the each activity in the whole dataset. Each class has almost equal number of data points so the dataset is balanced.

Now, let's analyze how many datapoints were collected from each subject. The figure 5 on 21 shows that more datapoints were recorded from subjects 1637 to 1640

and 1629 than the other subjects. Similarly, in gyroscope file, subject 1629 has the most entries. It appears more than 120k rows of the csv files. Whereas, the others almost appear for 70k rows.

Data Cleaning

The figures 7 and 8 on pages 22 and 23 respectively, shows the information such as the names of columns, number of data points and the data-type. You can note that the data-type of the values in column z-axis for both the sensors is given as an 'object'. It should be a float like other axes. We handle this error by converting the object data-type to a float64.

From the research paper, the author says that there are no null values throughout the dataset Weiss (2019). For our reference, we check the null values and drop if there are any for both the accelerometer and gyroscope files.

We combine both the CSV files into one and we lose some data because although the dataset is balanced for each class there is still variability while recording the signals from each subject. Therefore, we combine these two files using the 3 common identifiers: $subject\_ID, activity\_code, timestamp$. The number of datapoints reduces to 3 million from 7 million approximately.

Related Work

There are a lot of research going based on the given dataset. The research is about classifying all the 18 activities successfully. One of the researches by the creator of the dataset implements three types of models from the popular open source python's scikit-learn library Yoneda and Weiss (2017). They implement KNN, Decision Tree and Random Forest Classifiers with different parameter settings. The paper focuses on five combinations of the sensor data with 10 seconds sliding- window technique for pre-processing and feature extraction. According to their results, the Random Forest Classifier performed the best among all.

Walse, Dharaskar, and Thakare (2016) utilized the same but a smaller version of WISDM dataset with only 6 activities. The paper focuses on feature extraction

techniques to extract features such as Average, Standard Deviation, Average Absolute Difference and Time between Peaks. They developed models using WEKA Tool library and trained models LibSVM, J48 and Random Forest decision tree algorithms, IBK instance-based, J-RIP rule induction, Bagging and Logistic Regression with default setting. Through the experimental results, it can be noted that Random Forest Classifier had better performance. The overall accuracy for it was approximately 98% . However, the model still got confused between the activity 'walking' and 'standing'.

Feature Extraction

For feature extraction, we use two techniques: Principal Component Analysis (PCA) and Sequential Feature Selection Technique (Forward Selection) from scikit-learn libraries Pedregosa et al. (2011).

We applied these techniques to reduce the over-fitting (if any) of the models. The original dataset has total 6 input features: X, Y and Z axes of both the sensors. So, to find the best or important features for our application, we decide to reduce the number of columns and eventually reduce the complexity. The detailed information is in the below sub-sections.

Principal Component Analysis

We have total 6 inputs and after during the PCA, we will have at most 6 new uncorrelated features or Principal Components. But our main goal is to reduce the complexity as much as possible to reduce the over-fitting issues. We carry PCA in such a way that we end up having the first 3 PC (Principal Components). Then we fit the changes to our original input to transform it. Thus, we get a new set of features with reduced number of inputs that are uncorrelated to each other.

Sequential Feature Selection Technique

The other technique that we learnt during the Machine Learning class was forward and backward feature selection. This Sequential Feature Selector adds (forward selection) or removes (backward selection) features to form a feature subset in a greedy

fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator. This function is only available for the newest release of scikit learn library (0.24.2).

The parameter input for the function is the unfitted model, number of features to select and the direction of the selection (forward/backward). We use this method for Decision Tree Classifier to reduce the over-fitting issues. We fit it to the original X_train and y_train. After undergoing forward feature selection, we are left with 3 columns out of 6. It is a good technique because we can even check which columns are important by calling one of it attributes. The function returns an array containing boolean values as True for the columns that are important features and False for vice verse. We transform the set of inputs according to the result and then apply to the model.

Model Development

There are 18 output classes for our application. Thus, our problem can be solved using the classification models. We train around 6 different classification models excluding the fine-tuned models from scikit-learn, an open source library Buitinck et al. (2013). The sub-sections describes information about the implemented models. The next section will explain the enhancement or fine-tuning of these models.

Logistic Regression

We chose this model we an assumption that the activity output is linearly dependent on the input features. We used the default parameters except the *solver* for training. We selected $newton - cg$ as the *solver* parameter because it can handle loss for multi-class problems. Also, the default settings come with $L2$ penalty to avoid over-fitting issues.

Linear Discriminant Analysis

The LDA classifier is similar to Logistic Regression in terms of decision boundary. Here we assume that each class shares same covariance matrix and fits the Gaussian

distribution. We trained, validated and tested the classifier using the default parameters.

Quadratic Discriminant Analysis

After training the linear models, we found that the accuracy was low as the models were simple. So, we implement QDA to increase the complexity of the model and assume a non-linear quadratic decision boundary. Here also, we use the default parameters for training our model.

K-Nearest Neighbors

The QDA model also had a lower accuracy for both train and test sets. Therefore, we implement are more complex model i.e KNN. This model also follows the default settings while training. The default settings include the neighbors $n = 5$. KNN with $n = 5$ has a very high flexibility. We noted higher accuracy in terms of training, validation and test sets.

Decision Tree

We even train tree based models for our application. We take the idea of implementing these models from the research paper that we referred. Also, the decision trees are non-linear but simpler and can be trained in order to learn simple decision rules. We train the default Decision Tree Classifier with one of the most important parameters for classification: $GiniIndex$.

Random Forest Classifier

The research papers concluded that the Random Forest Classifier was a better model in terms of activity recognition application. We implement this classifier to verify their results. Random Forest Classifiers are usually implemented to overcome over-fitting issues in the Decision Trees. We apply the default parameters except the $n_estimators$. The default settings already better according to our knowledge from the

course. The important parameters for the classifier are *Gini Criterion*, $max\_features$, *bootstrap*.

Fine-tuning of Models

The previous section explained the base models that we implemented with the default settings. After checking the train, validation and test results using those parameters, we modify a few parameters and note the effects by keeping other parameters constant. In a nutshell, if the model suffers from over-fitting, we make it simpler, if model has lower accuracy, we add flexibility to it, etc. The below subsections are divided as: $Linear\ Models$, $Non-linear\ Models$ and $Tree-based\ Models$.

Linear Models

We group the *Logistic Regression* and LDA as one, because both of them has linear decision boundaries. Both models had almost the same training, validation and test accuracy. The accuracy is approximately 24 percent which is extremely low. This gives a clue that the model can be still improved if we add non-linearity and complexity to it .

We create a new set of features with non-linearity. We multiply the values $X-axis$ of Accelerometer with the $X-axis$ of Gyroscope and name the column as x. This is done to Y and Z axes too. The figure 9 on page 23, displays the addition of 3 more columns.

We train the *Logistic Regression* model with these new features and note that there is no significant improvement in the accuracy. The accuracy only increases by 1%. If we had more non-linear/linear features/columns we would have got better accuracy. Further, we do not train the LDA because of limited time and we argue that it would also give the same result.

Non-linear Models

For simplicity in drawing results and comparison, we group QDA and KNN as our non-linear models. However, in general all the models except LDA and *Logistic*

Regression have non-linear decision boundaries and can be counted as non-linear models.

We implement the QDA model using default settings. The accuracy are better than LDA and *Logistic Regression* but only 5%. And after looking at the results of non-linear features we decide to implement a more complex model than QDA. Therefore, we switch to KNN with default settings.

The KNN with $n = 5$ is the complex model till now. We faced some over-fitting issues as the gap between the training accuracy and validation/test accuracy is large. So, for fine-tuning this model, we opt for a value of n greater than 5. As the value of n increases the flexibility of the KNN model decreases which may lead to reducing the over-fitting problem. With this, the we selected $n = 25$ and trained the model. There was a notable decrease in over-fitting with some loss in accuracy value. This model is lesser complex than $n = 5$ but more complex than QDA. Thus, our assumption was correct about QDA. The KNN with $n = 25$ neighbors is better for our application with a limitation of lower accuracy.

Tree-based Models

In the $Tree - based$ technique we group simple $Decision\ Tree\ Classifier$ and $Random\ Forest\ Classifier$ for comparison. As discussed earlier, we apply default parameters to both the models while training. For pruning the trees, we apply different feature selection or elimination methods to both the models and acquire the results.

Precisely, the *Decision Trees* with default parameters is a *Recursive Binary Splitting* (RBS) technique. And due to that, our model suffers from a high over-fitting problem. The training accuracy is almost 100% with extremely lower validation and test performance. We could try and change the parameters and come up with a better tree but this becomes a very tedious task. To reduce the features, we use one of the feature extraction techniques mentioned in the section above. We reduce the number of columns by *Sequential Feature Selector* method and train the model again with a few changes in parameters. We overcome the over-fitting problem by reducing the columns,

decreasing the maximum depth of the tree and criterion for mis-classification error.

Similarly, the default parameters for *Random Forest Classifier* lead us to over-fitting issues. However, the over-fitting was lower than the simple *Decision Tree* implemented above. To overcome that, we used the new set of uncorrelated features generated by *PCA*. We got some serious reduction in over-fitting however with lower accuracy values for training, validation and test.

Performance

This section goes over the performance for each model or sub-model that we implemented. We have around 3 million data-points therefore we divide the dataset using *Validation Set Approach*. We fix the random state when splitting to a fixed value to lower the variability during the splitting of the data. We calculate the training, validation and test accuracy for each model. We also define performance metrics such as *precision*, *recall* and $F1 - score$ for every activity for each model.

The table 1 on page 17 shows the accuracy results of linear models. The description column in the table shows the features that were used to train the model. Original means the given dataset and new features implies the non-linear features. As you can note that the accuracy remains around 25%. There is no over-fitting which means that the dataset requires a more complex model. The figure 10 on page 24 shows the confusion matrix of the *LDA* classifier. We only included *LDA*'s confusion matrix in the report because we record highest accuracy among the linear models. The classifier is able to classify the activities like *standing*, *typing*, *brushing teeth* and *writing* more accurately. However, it still confuses *typing* activity as *sitting*.

We now compare the *LDA* with *QDA*. The overall accuracy increases from 25% approximately to 31% when we change the decision boundary from linear to quadratic. The precision and recall values also increased for *standing* activity. Also, the linear models were not able detect the *eating sandwich* activity at all but *QDA* is able to detect it.

The *KNN* model with $n = 5$ has over-fitting problem. The training accuracy is

65.41% whereas validation is 52.57%. So, we train a simpler model using $n = 25$ neighbors. The training accuracy reduces to 56.84% and validation accuracy increases to 53.73%. The figures 11 and 12 on pages 25 and 25 shows the $precisin$, $recall$ and $F1 - scores$ for KNN with $n = 5$ and $n - 25$ neighbors respectively. You may notice that the $F1 - score$ slightly improves for $n = 25$ model.

The $Decision\ Tree$ model also suffers from over-fitting. We prune the tree by eliminating some of the features. The table 2 on page 18, shows the training, validation and test accuracy. The RBS without feature selection and default parameters shows a lot of over-fitting. The accuracy of $RBS + ForwardSelection$ is very low but it is due to the parameters passed in the decision tree model while training. This model can still be improved by changing the parameters of the decision trees and making it complex.

The same conclusion can be drawn for $Random\ Forest\ Classifiers$ with and without PCA. Due to constraints on the tree, the classifier with PCA shows reduced accuracy but can be improved by changing the flexibility. If you compare the confusion matrices for both the models in figures 13 and 14 on pages 26 and 27 respectively, you see that the classifier is able to classify almost all the activities correctly. However, the performance of the PCA is lower but can be improved by changing the classifier's parameters.

Discussions and Conclusions

Activity recognition can be very crucial in terms of health monitoring. To recognize the activities properly we require an efficient model. We implemented around 6 unique models excluding the fine-tuned versions. We understand the effects on the performance and accuracy by addition or elimination of the complexity in the models. We compare various performance metrics to finally get and optimum model for our application. According to our training and testing results, we conclude that KNN with $n = 25$ neighbors performed better. We record 50% highest accuracy for KNN and it can be still improved by changing the values of n and other parameters of the model. However, the $Tree - based\ models$ can also be used with a few modification in the

parameters.

We mainly faced issues when reading the *.txt* files into *.csv* files. The initial conversion was challenging but our professor and a few online resources such as $GeeksForGeeks$ and $StackOverflow$ helped us a lot. We even referred to a few notebooks on the $Kaggle$ website. The main code cites the reference code wherever we used. The datapoints for our dataset were around 3 million so the training and validation took a lot of time. We used Google Colab with TPU /GPU acceleration because of our physical machines lack the computation requirements. We faced run-time and low Disk space availability issues though training on Colab using TPU. The models implemented in the main code are the best models that we could finally keep and interpret the results. But we actually trained and tested more models with different parameter settings.

Acknowledgements

We would like to thank our professor Birsen Sirkeci for her guidance and support throughout the project.

References

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., ... Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *Ecml pkdd workshop: Languages for data mining and machine learning* (pp. 108–122).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Walse, K. H., Dharaskar, R. V., & Thakare, V. M. (2016). Performance evaluation of classifiers on wisdm dataset for human activity recognition. In *Proceedings of the second international conference on information and communication technology for competitive strategies.* New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2905055.2905232` doi: 10.1145/2905055.2905232

Weiss, G. (2019, September). *Wisdm smartphone and smartwatch activity and biometrics dataset data set.* (https://archive.ics.uci.edu/ml/datasets/WISDM+Smartphone+and+Smartwatch+Activity+an

Yoneda, K., & Weiss, G. (2017, 10). Mobile sensor-based biometrics using common daily activities. In (p. 584-590). doi: 10.1109/UEMCON.2017.8249001

Table 1

Accuracy results for models with linear decision boundary

Model	**Description**	**Training**	**Validation**	**Test**
Logistic Regression	Original	24.09%	25.03%	24.93%
Logistic Regression	Original + New Features	25.20%	25.32%	25.20%
LDA	Original	25.47%	25.58%	25.43%

Table 2

Accuracy for Simple Decision Trees

Model	**Training**	**Validation**	**Test**
RBS	99.98%	46.44%	46.34%
RBS + Forward Selection	36.16%	36.13%	35.98%

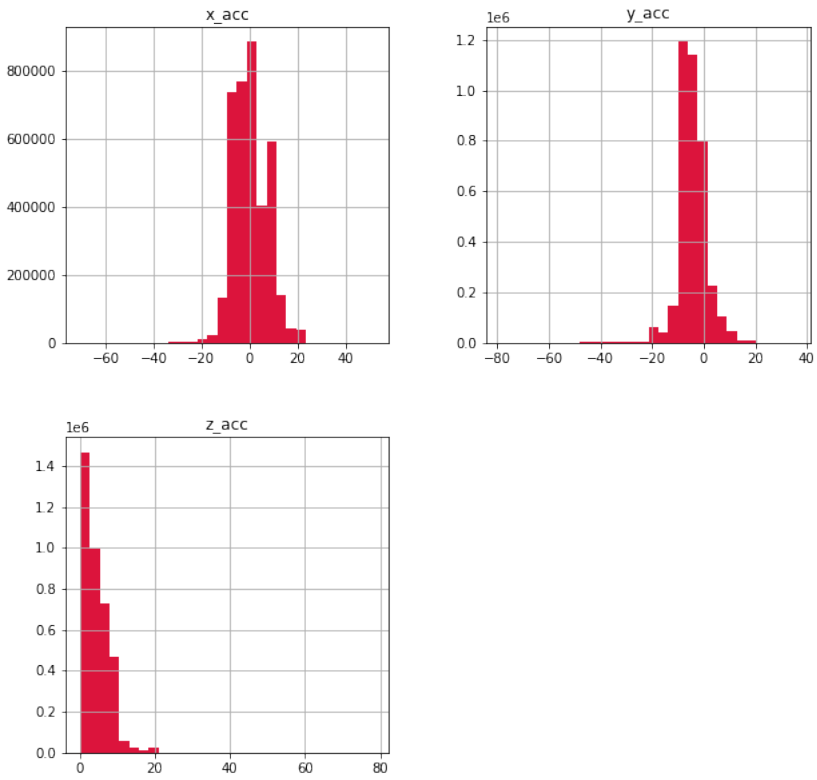

Figure 1. Histogram plot: Accelerometer Input

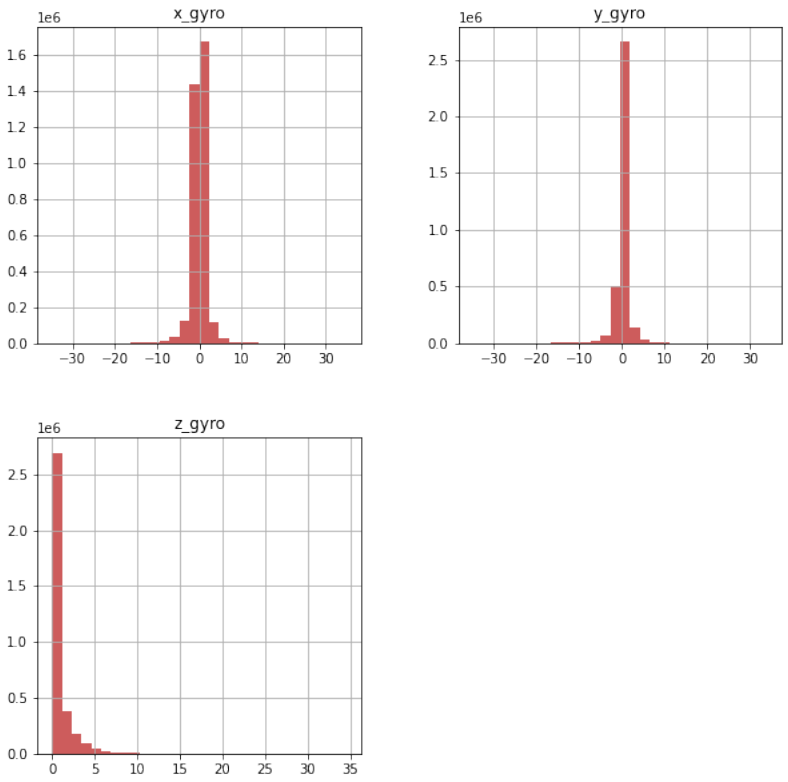

Figure 2. Histogram plot: Gyroscope Input

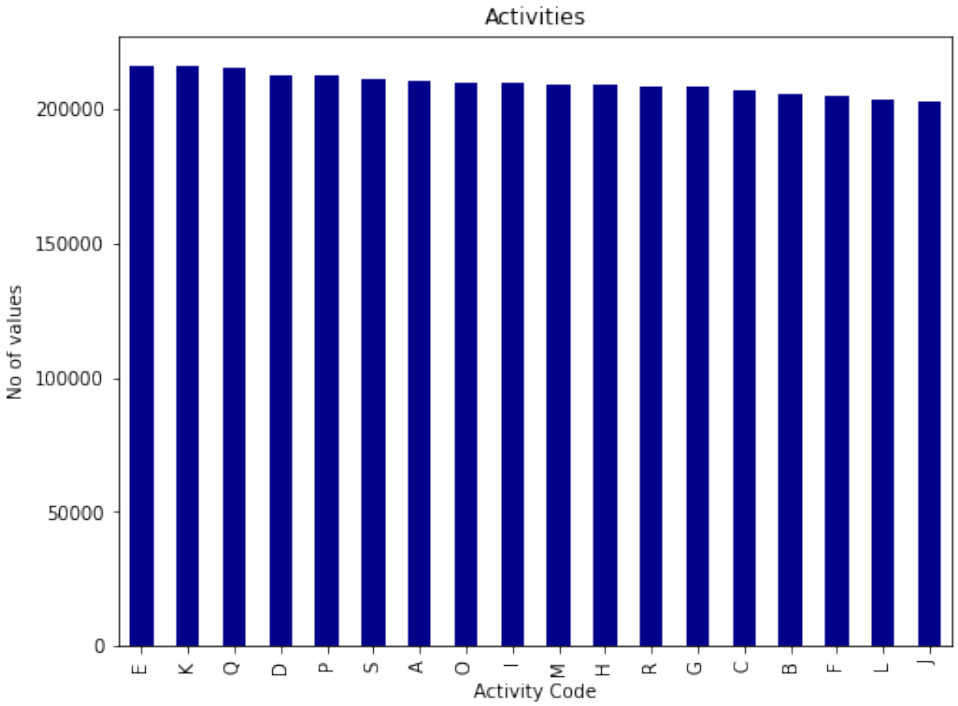

Figure 3. Counts of each Activity recorded by Accelerometer

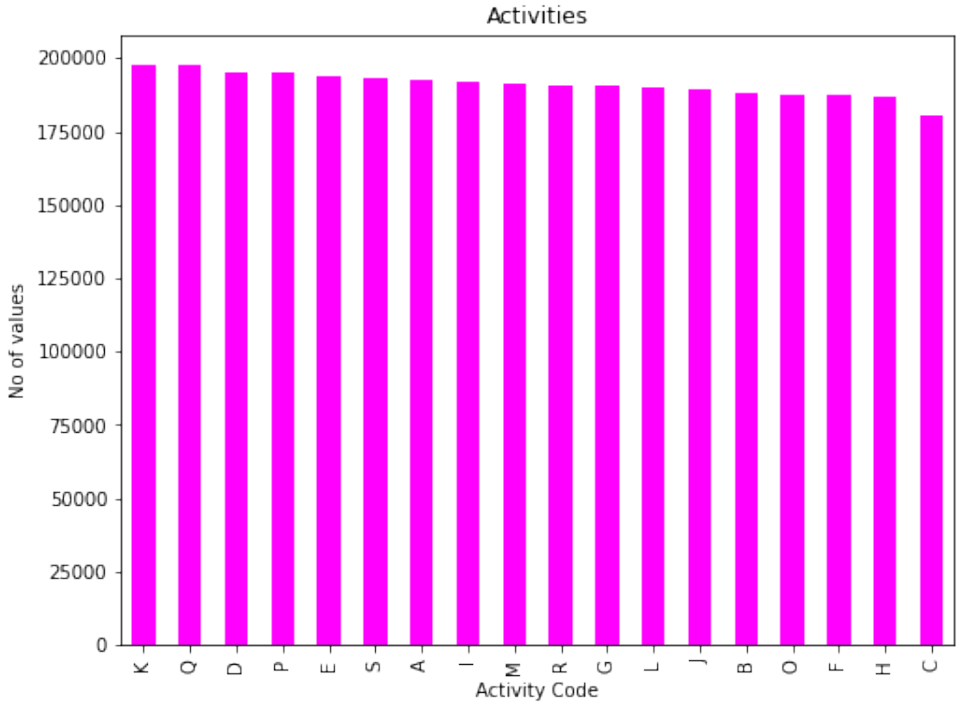

Figure 4. Counts of each Activity recorded by Gyroscope

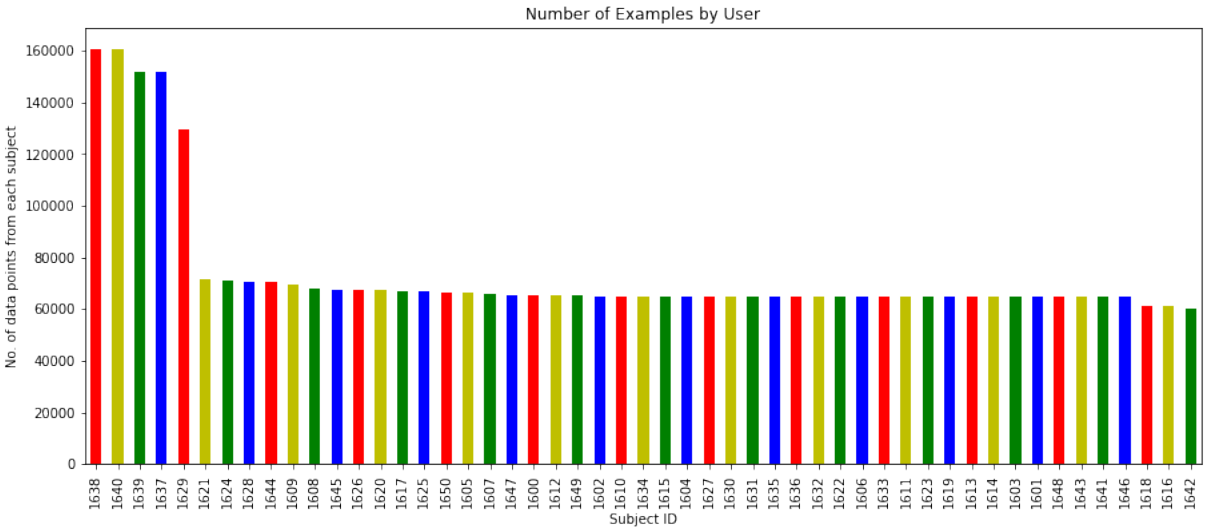

Figure 5. Datapoint entry for each subject in Accelerometer file

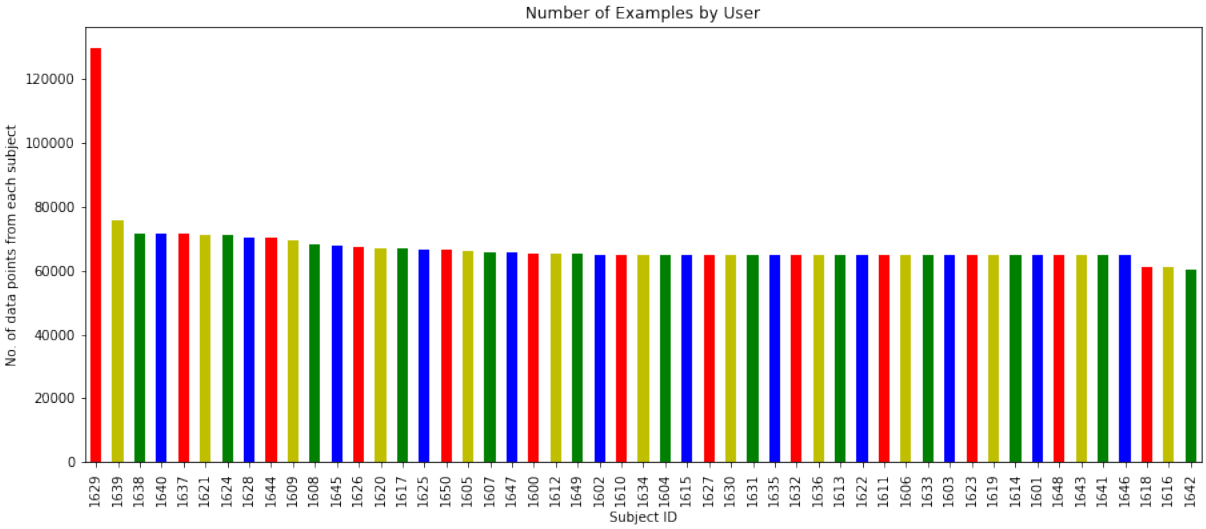

Figure 6. Datapoint entry for each subject in Gyroscope file

```
1   acc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3777046 entries, 0 to 3777045
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   subject_ID     int64
 1   activity_code  object
 2   timestamp      int64
 3   x_acc          float64
 4   y_acc          float64
 5   z_acc          object
dtypes: float64(2), int64(2), object(2)
memory usage: 172.9+ MB
```

Figure 7. Accelerometer file information

```
1   gyro.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3440342 entries, 0 to 3440341
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   subject_ID     int64
 1   activity_code  object
 2   timestamp      int64
 3   x_gyro         float64
 4   y_gyro         float64
 5   z_gyro         object
dtypes: float64(2), int64(2), object(2)
memory usage: 157.5+ MB
```

Figure 8. Gyroscope file information

	x_acc	y_acc	z_acc	x_gyro	y_gyro	z_gyro	x	y	z
0	7.181558	-0.772878	0.583437	1.975182	0.124283	3.194493	14.184886	-0.096055	1.863785
1	5.500828	1.934964	0.176423	1.472378	0.404447	2.381696	8.099297	0.782591	0.420185
2	5.184793	2.409016	0.181211	0.345328	0.300051	1.506048	1.790453	0.722828	0.272913
3	7.191135	0.541539	0.200365	-1.170543	-0.169730	0.205831	-8.417535	-0.091916	0.041241
4	12.685827	-0.317980	1.253813	-1.899184	-0.529790	3.083111	-24.092723	0.168463	3.865646

Figure 9. Adding New Columns/Features

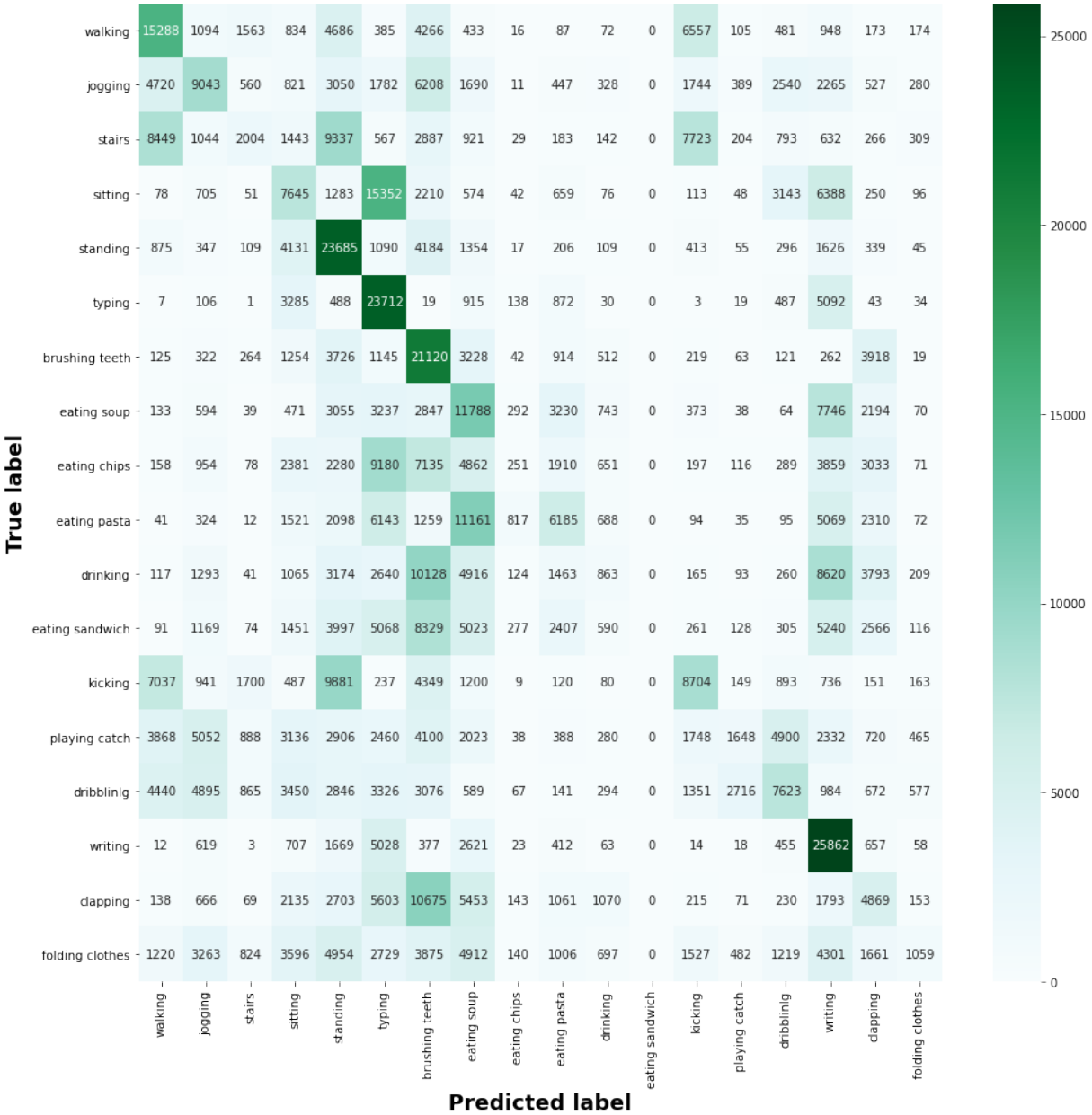

Figure 10. Confusion Matrix for LDA Classifier

	precision	recall	f1-score	support
walking	0.45	0.61	0.52	29627
jogging	0.55	0.63	0.58	28844
stairs	0.36	0.37	0.36	29408
sitting	0.70	0.78	0.74	30908
standing	0.76	0.84	0.80	31259
typing	0.60	0.72	0.65	28543
brushing teeth	0.51	0.62	0.56	29782
eating soup	0.41	0.47	0.44	29614
eating chips	0.40	0.36	0.38	29687
eating pasta	0.44	0.42	0.43	30146
drinking	0.56	0.57	0.56	31333
eating sandwich	0.51	0.40	0.45	29596
kicking	0.41	0.31	0.35	29538
playing catch	0.41	0.31	0.35	29159
dribblinlg	0.57	0.50	0.53	30090
writing	0.70	0.70	0.70	31306
clapping	0.61	0.59	0.60	29936
folding clothes	0.35	0.25	0.29	30191
accuracy			0.53	538967
macro avg	0.52	0.52	0.52	538967
weighted avg	0.52	0.53	0.52	538967

Validation Accuracy: 52.567411362847814%

Figure 11. Performance metrics for KNN with 5 neighbors

	precision	recall	f1-score	support
walking	0.52	0.61	0.56	29627
jogging	0.62	0.63	0.62	28844
stairs	0.42	0.35	0.38	29408
sitting	0.74	0.75	0.75	30908
standing	0.77	0.83	0.80	31259
typing	0.60	0.73	0.66	28543
brushing teeth	0.50	0.64	0.56	29782
eating soup	0.42	0.44	0.43	29614
eating chips	0.42	0.32	0.37	29687
eating pasta	0.43	0.45	0.44	30146
drinking	0.52	0.57	0.55	31333
eating sandwich	0.50	0.39	0.44	29596
kicking	0.42	0.36	0.39	29538
playing catch	0.44	0.28	0.34	29159
dribblinlg	0.58	0.55	0.56	30090
writing	0.66	0.76	0.70	31306
clapping	0.57	0.64	0.61	29936
folding clothes	0.35	0.31	0.33	30191
accuracy			0.54	538967
macro avg	0.53	0.54	0.53	538967
weighted avg	0.53	0.54	0.53	538967

Validation Accuracy: 53.72536723027569%

Figure 12. Performance metrics for KNN with 25 neighbors

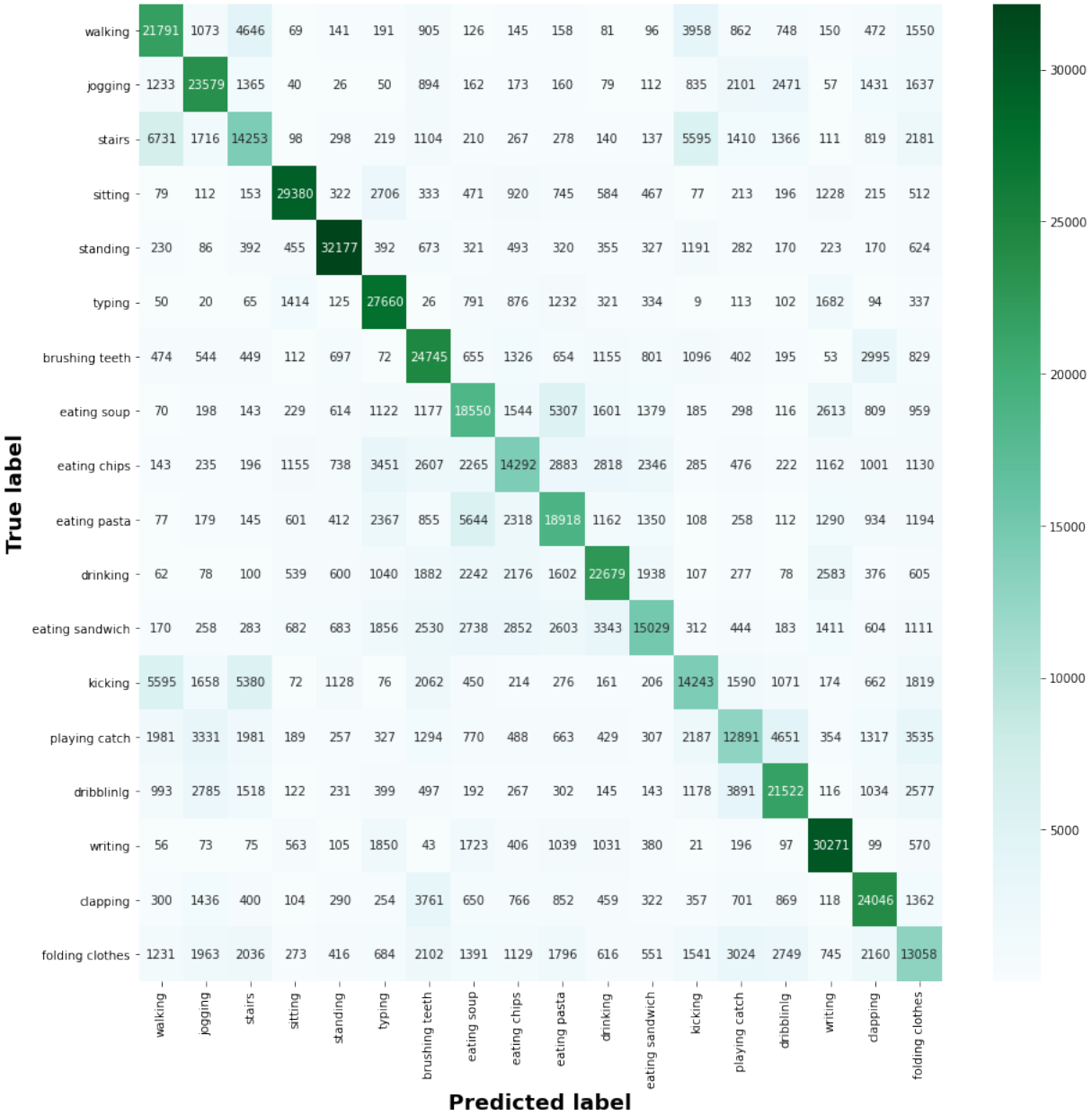

Figure 13. Confusion Matrix for Random Forest Classifier

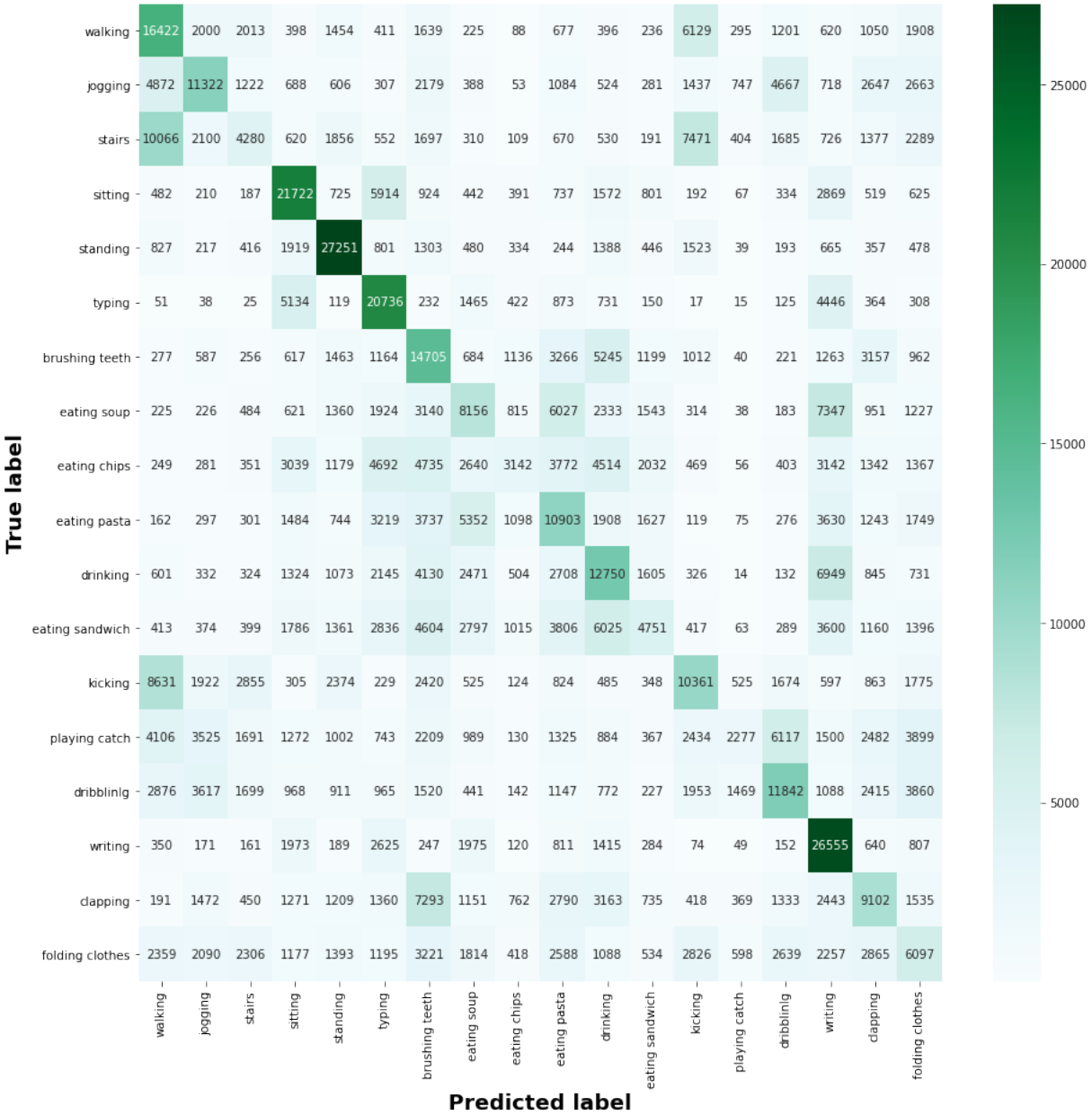

Figure 14. Confusion Matrix for Random Forest Classifier + PCA