FYS-STK4155 - PROJECT 3: CLASSIFYING HUMAN ACTIVITY USING TREE ENSEMBLES

ALEXANDER HAROLD SEXTON
ERIK JOHANNES HUSOM

DECEMBER 9, 2019

Abstract. In this project we use tree-based and ensemble machine learning methods, specifically decision tree, bagging, random forest, AdaBoost, gradient boosting and XGBoost, on a human activity recognition (HAR) classification problem. The dataset consists of raw data from a triaxial, chest-mounted accelerometer, and was collected from 15 subjects performing 7 different activites. The goal is to use these methods to recognize what activity a subject performed based on the accelerometer data. When using a training set containing data from all 15 subjects (setting 1), all methods gave a test accuracy score above 97%, and the highest score, 99.5%, was obtained by both gradient boosting and XGBoost. When using separate subjects in the training and test set (setting 2), the highest score obtained was 19.5%, when using Gradient Boosting. The parameters of the models were tuned by using a cross-validation grid search, but overfitting of the training data proved to be a challenge. Improvement was made by reducing the depths of the trees, and XGBoost scored 60.5% in setting 2 when using a depth of 1. A simplified target set, with 4 activity categories, was also explored.

Contents

1. Introduction

Human activity recognition (HAR) is technology that aims to recognize the activity that a person performs, based on observations of the person. Usually these observations are obtained by collecting data from various sensors placed on different parts of the body. This has many applications, for example in fields such as healthcare and sports science, and it is also widely used through the ubiquity of smartphones, smartwatches and other wearables that constantly track the movements of the user. One of the most common cases is recreational users who want to track their everyday activity, but this technology has also more critical applications, such as fall detection for sick and elderly[1].

Activity recognition can be seen as a typical classification problem, with activity type as the target variable. While there are many possibilities with regards to what type of data to use as features of a model, accelerometer data are arguably one of the most attractive data types; accelerometers are easy to wear, cost-efficent, records data closely connected to the movements of a person, and are already present in most smartphones and wearables. In this project we deal with a dataset publicly available at the Machine Learning Repository of University of California, Irvine (UCI)[1], which provides data from a single chest-mounted accelerometer. Since activity sensors produce time series data, the predicted target for each observation will also be dependent on several of the preceeding data points. Many common machine learning methods, including the ones we explore in this project, will then require that we do a feature extraction on "windows" of the time series data, which has been done with success in previous research[2]. Both the feature extraction and the training/test-split of the dataset have significant effect on the model performance, and this is one of the challenges we face in this project.

In this report we present the background theory and the methods used in the project. We have chosen to focus on tree-based methods, ranging from a single decision tree through bagging, random forests and several boosting algorithms, with trees as the base classifier. Our findings are presented in the results section, where we also include results from related research[3] for the sake of comparison. This is followed by a discussion of the results, and lastly a conclusion. The appendix contains results from a deeper analysis of the dataset; these findings were obtained in the last stages of the project, where we explore additional methods to deal with overfitting of the training data.

2. Methods

2.1. **Decision Trees.** Decision trees is a powerful machine learning algorithm capable of fitting complex data sets, used both for classification and regression. The structure of a decision tree is much like a real life tree, and consists of nodes, branches and leaves, where a node represents a test on a descriptive feature in the data, the branch represents the outcome of this test, and the leaves represent an outcome or a target feature. The main idea is to find the descriptive features in the data which contain the most information about the target feature, and then split the data set along these values such that the feature values of the underlying data set are as pure as possible. The most common measures of the impurity of a node is the *gini index*[4][p. 180]:

$$g_m = 1 - \sum_{k=1}^{K} p_{m,k}^2 \tag{1}$$

[1]Link to the dataset at UCI: https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

and the *information entropy*[4][p. 184]:

$$s_m = -\sum_{k=1}^{K} p_{m,k} \log p_{m,k}. \tag{2}$$

where p is the ratio of class k instances among the training instances in node m. A high value of either of these measures would represent a node which contains little information about which class the observation belongs to, and conversely a low or zero value represents a node with only one outcome, resulting in a leaf node.

In building a tree, the *Classification and Regression Tree* (CART) algorithm is commonly used, which splits the training set into two subsets using a single feature k and threshold t_k. This pair (k, t_k) is found by minimizing the cost function given by[4][p. 182]:

$$C(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \tag{3}$$

where $G_{left/right}$ measures the impurity(eq. 1 or 2) of the left and right subset and $m_{left/right}$ is the number of instances in the left and right subsets. Once a split has been made, the procedure recursively repeats for the subsets until the impurity reaches zero. This strategy by itself will likely lead to a very overfit model which does not generalize, so it is common to specify either a maximum tree depth or a minimum impurity threshold which should result in a leaf node.

Since the CART algorothm uses node impurity for node splitting, this measure also tells us something about how important a given feature is to determining the outcome of an observation, which can give additional insight into the problem at hand by allowing us to visualize the tree in terms of how it makes predictions. Feature importance is also possible to extract when using ensemble learning methods, which are discussed in the section below. This is done by taking a weighted average, across all the trees in the forest, of how much a given descriptive features reduces node impurity.

2.2. **Bagging and Random Forests.**

2.2.1. *Bootstrap Aggregation.* Bootstrap aggregation (or *bagging*) is an ensemble learning method based on aggregating many different predictive models. In bagging, the individual models are trained on random subsets of the data which are drawn with replacement, and a prediction is made by obtaining the prediction of each individual model, then predict the class which gets the most votes. This method of machine learning is known to reduce both the bias and variance, and as a result yield a much higher prediction accuracy than a strong classifier by itself. The parameters which are normally tuned in training a bagging model is the number of subsamples which are drawn in each bootstrap, and how many individual predictive models to include in the final aggregated model.

2.2.2. *Random Forests.* The random forest algorithm is another way of aggregating predictive models, but unlike bagging, it restricts itself to only using decision trees. The strength of random forests lies in that it introduces extra randomness when growing trees, by only searching for the best feature among a subset of features when growing a tree. In each splitting of a node, a fresh sample of typically $m \approx \sqrt{p}$ is randomly selected, which results in greater tree diversity and yielding an overall better model with lower variance. As with a simple tree, the maximum depth of the trees should be tuned to avoid overfitting, as well as the total number of trees in the ensemble.

In this project, we use Scikit Learn's implementation of decision trees, bagging and random forests.

2.3. **Boosting.** Boosting methods are based on the idea that we combine several weak classifiers into one strong classifier, as is the case with bagging, but with boosting the classifiers are made sequentially. When using boosting, we train the classifiers one after the other, where each new classifier is trying to learn from the errors of the preceding one. In this project we will only use decision trees as the base estimator for our boosting algorithms. We have chosen to use three different boosting methods, and a high-level description of these are presented below.

2.3.1. *AdaBoost.* The AdaBoost (adaptive boosting) is one of the most common boosting algorithms used in machine learning. For each boosting iteration, we produce a weak classifier $G_m, m = 1, 2, ..., M$, where M is the number of classifiers. Using a binary classifier as an example, where the output is $y \in [-1, 1]$, we combine the predictions by using a weighted majority vote for our final classifier G[5]:

$$G = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m\right). \tag{4}$$

The variables α_m are weights for each of the classifiers G_m, and are computed by the AdaBoost algorithm. The training data points $(x_i, y_i), i = 1, 2, ..., N$ are modified for each iteration by applying weights $w_1, w_2, ..., w_N$, which means that each individual observation is modified. The initial weights are set to $w_i = 1/N$, where N is the number of observations. Then, for each iteration, the weights of correctly classified observations are decreased, and the weights of misclassified observations are increased, such that each successive classifier are more affected by the observations that its predecessor failed to classify. In more general terms, the hypothesis $f(x)$ can be expressed as[5][p. 341]

$$f(x) = \sum_{m-1}^{M} \alpha_m b_m(x; \gamma), \tag{5}$$

where b_m are elementary basis functions of x, which also depend on a number of parameters γ_m. In our case this will be the weak classifiers that are combined into our final model. Both AdaBoost and the other boosting methods takes a parameter called the learning rate, which affects how much the weights are adjusted for each iteration.

We use Scikit-Learn's implementation of AdaBoost[2] in this project, and since we are predicting more than two classes, Scikit-Learn will use a multiclass version of AdaBoost that is called SAMME (*Stagewise Additive Modeling using a Multiclass Exponential loss function*).

2.3.2. *Gradient Boosting and XGBoost.* Gradient boosting is another boosting method that combines weak classifiers into a strong one. While AdaBoost uses weighted data points to adjust each subsequent classifier, gradient boosting uses the residuals from each classifier to improve predictions. More generally, for each iteration m we fit a tree, our base learner, to the negative gradient values of the loss function[5][p. 361]:

[2]Link to the Scikit-Learn implementation of AdaBoost: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier`

$$r_{i,m} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}, \tag{6}$$

where $r_{i,m}$ are the negative gradient values, L is the loss function, and i is the index of the observations. The negative gradient fit is then added to our estimate $f_m(x)$, and this is repeated for every iteration until we are left with our final classifier $f_M(x)$. With Scikit-Learn's implementation `GradientBoostingClassifier`, we use the default loss function *deviance*, the multinomial log-likelihood loss function.

XGBoost (Extreme Gradient Boosting) is a gradient boosting library, that provides a highly efficient way to use tree boosting on machine learning problems. The algorithms used in XGBoost are similar to those of gradient boosting, but are optimized for even higher performance. The details can be reviewed in the original XGBoost article[6].

2.3.3. *Boosting parameters.* For boosting methods there are typically three parameters we want to tune in order to optimize the model[7][p. 322]:

- The number of trees. In the code this is called `n_estimators`, and controls how many boosting iterations we run. With to many trees, the model might overfit to the training data.
- The learning rate, a small positive number. This parameter affects how fast the boosting learns.
- The depth of the trees, in other words the complexity of each tree in the ensemble. In the code, this parameter is passed as `max_depth` to the functions of Scikit-Learn.

These three parameters are tuned using grid search and cross-validation, with Scikit-Learn's method `GridSearchCV`.

2.4. **Description of the Data.** The data set[3] consists of measurements collected from a wearable accelerometer mounted on the chest of 15 participants performing 7 activites. The features of the raw data are the uncalibrated acceletations in the x, y and z directions, sampeled at a frequency of 52 Hz, totalling 1.9×10^6 measurements. The target features are labeled from 1 to 7, with each number corresponding to the following activities:

(1) Working at computer
(2) Standing up, walking and going up/down stairs
(3) Standing
(4) Walking
(5) Going up/down stairs
(6) Walking and talking with someone
(7) Talking while standing.

When preprocessing this data, we extract features from the raw acceleration measurements using a window size of 52 samples (corersponding to 1 second) with 26 samples overlapping between consecutive windows[4]. From the samples in each of these windows, we calculate and form in total 16 descriptive features, shown in table 2.1. When training our models, we have chosen two cases of splitting the data

[3]The data set is available at the UCI website: https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

[4]Overlapping between consecutive windows has demonstrated good results in previous work [8].

TABLE 2.1. Features extracted from the raw data.

Index	Feature	Index	Feature
0	Mean x-acceleration	8	Range of z-acceleration
1	Mean y-acceleration	9	Magnitude of acceleration
2	Mean z-acceleration	10	Velocity in x-direction
3	Standard devation of x-acceleration	11	Velocity in y-direction
4	Standard devation of y-acceleration	12	Velocity in z-direction
5	Standard devation of z-acceleration	13	Periodicity of x-acceleration
6	Range of x-acceleration	14	Periodicity of y-acceleration
7	Range of y-acceleration	15	Periodicity of z-acceleration

into training and test. In the first case, the activity data from all the subjects are randomly shuffeled, and 80% are then used for training and validation, and 20% for testing. In the second case, we use only measurements from 12 of the subjects for training and validation, and the remaining 3 for testing. In the rest of the report, these cases are reffered to as *setting 1* and *setting 2*.

2.5. **Source code.** The source code of this project is written in Python, and can be found in the GitHub repository at `https://github.com/ejhusom/FYS-STK4155-project-3/`. The repository contains our source code in the folder `src`, which consists of the following files:

- `ActivityData.py`: Preprocessing of the activity data, including loading, feature extraction and scaling.
- `Boosting.py`: Functions for using AdaBoost, gradient boosting and XGBoost on the activity data, and analyzing the performance of these methods.
- `Trees.py`: Functions for using a single decision tree, random forest and bagging on the activity data, and analyize the performance of these methods.

3. RESULTS

To find the optimal parameters for our machine learning methods, we performed a cross-validation grid search. The results are presented in table 3.1. For the three boosting methods the following ranges were tested for each parameter:

- Estimators: 100, 150, 200.
- Max depth: 5, 7, 9, 11.
- Learning rate: 1, 0.5, 0.1.

For the bagging and random forest methods we did a search over the following parameter space:

- Estimators: 100, 150, 200, 250, 300.
- Max depth: 3, 5, 7, 9, 11, 13, 15.

In the case of the simple decision tree, we did a tree depth search from 2 to 20. We have also used both the gini index and information entropy as node impurity measures for the trees, in which the information entropy gave the best scores in every case.

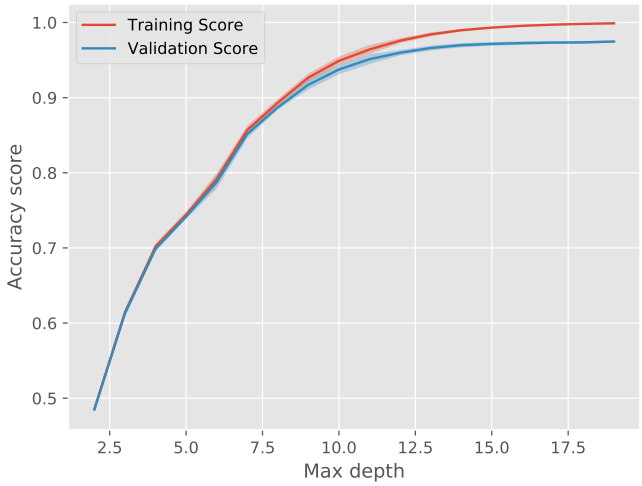

FIGURE 3.1. Validation curve of a simple decision tree on the setting 1 data. Optimal tree depth is 19, with validation score 97.5 and test score 97.8.

Figure 3.1 shows the validation curve of the simple decision tree, in which we found that a depth of 19 was optimal, giving an accuracy score of 97.8 on the test data. We see from the figure that a tree with only depth 2 has roughly a 50% accuracy of predicting the correct class among the 7, with an increasing trend until it reaches a tree depth of approximately 13, after which both the validation and training scores remain constant.

Table 3.2 shows the accuracy of our ensemble methods, with their optimal parameter configurations listed in table 3.1. We see that gradient boosting and XGBoost have the best performance with an accuracy score of 99.5, while AdaBoost and random forests perform almost equally as well with accuracy scores of 99.1. The bagging classifier is only slightly worse with a score of 97.8. Using setting 2, gradient boosting performs best with a score of 19.5, while the rest of the classifiers are below 14.

In table 3.3 we present the results from a similar research paper on human activity recognition from Ravi et al.[3]. This paper compared several machine learning methods, and we have included the ones that matches the methods of our project. Ravi et al. used raw data from a triaxial accelerometer, akin to us, and extracted a similar feature set to ours, by using window sampling to calculate mean, standard deviation etc. The sampling window was however 5.12 seconds, while ours was 1 second, and they extracted some additional features (details can be found in the original research paper[3]). Also note that setting 2 is defined slightly different in this case, as mentioned in the table caption. For setting 1, the decision tree outperformed bagging and boosting with a score of 98.53, while bagging gave the best performance in setting 2, with a score of 63.33.

Figure 3.2 shows the confusion matrices for the best performing classifiers in both setting 1 and 2. The feature importances are shown in figure 3.3, estimated for all the methods we tested in this project. The overall best performing classifier, gradient boosting, is highlighted in red. Feature 4, standard deviation of y-acceleration, stands out as the most significant feature in most methods, while 5-8 (standard deviation of z-acceleration and ranges of all axes) are generally regarded as relatively unimportant.

TABLE 3.1. Configuration of the classifiers for the two different cases, found by performing a cross-validation grid search on the parameters.

Classifier	Setting 1			Setting 2		
	Estimators	Max depth	Learning rate	Estimators	Max depth	Learning rate
Decision tree	1	19	N/A	1	17	N/A
Bagging	200	15	N/A	200	15	N/A
Random forest	200	15	N/A	300	15	N/A
AdaBoost	150	11	1	200	11	1
Gradient boosting	200	7	0.1	200	7	0.1
XGBoost	200	5	0.5	200	7	0.5

,

TABLE 3.2. Accuracy of classifiers for the two different settings.

Classifier	Accuracy (%)	
	Setting 1	Setting 2
Decision tree	97.8	7.2
Bagging	98.9	12.0
Random forest	99.1	10.9
AdaBoost	99.1	13.5
Gradient boosting	99.5	19.5
XGBoost	99.5	13.6

TABLE 3.3. Results from the research paper by Ravi et al.[3]. Note that in this case setting 2 consisted only of data from one subject, while the test data was from another subject (setting 1 and 2 corresponds to respectively setting 2 and 4 in the paper by Ravi et al.).

Classifier	Accuracy (%)	
	Setting 1	Setting 2
Decision tree	98.53	57.00
Bagging	95.22	63.33
Boosting	98.35	57.00

4. DISCUSSION

For setting 1 (mixed subjects) we found that all of our classifiers performed almost perfectly, with accuracy scores in the 99-th percentile on the test data in almost all of the cases (table 3.2). The single decision tree performed almost equally as well as the ensemble and boosting methods, which at first thought is quite surprising, though it's configuration is of a high complexity, with a tree depth of 19. We achieve high accuracy scores with all of our classifiers, but the parameter configurations that were found to be best, were in most cases an edge case of the grid search. Due to the high computational cost of further reducing the error, we have opted to not expand the grid search.

Although there is very low variance in the error of our models for the mixed subjects case, there is good reason to suspect that the models are overfitted to the movement patterns of these 15 subjects, due to the training data being very similar to the test data. This suspicion is very much supported by our results when training and testing the models on separate subjects, where we from table 3.2 see that the

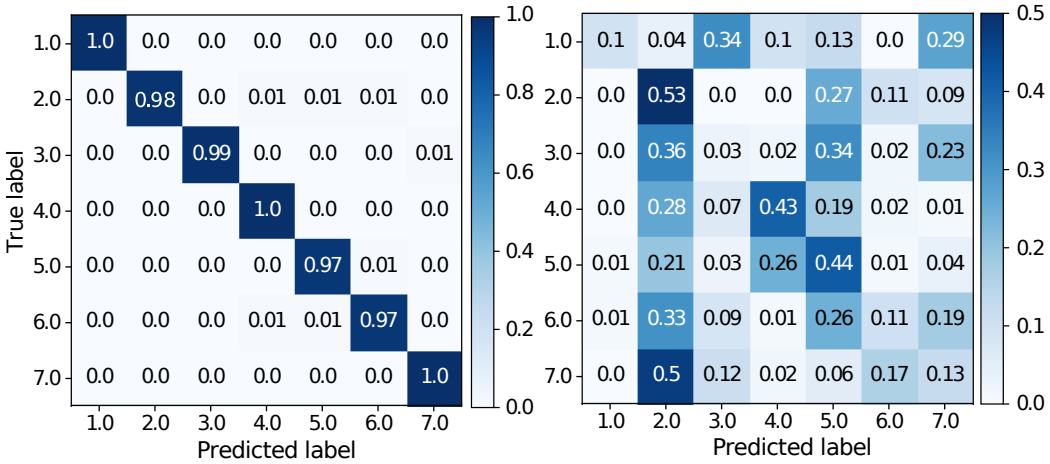

FIGURE 3.2. Confusion matrices. Left: Setting 1, using XGBoost. Right: Setting 2, using gradient boosting.

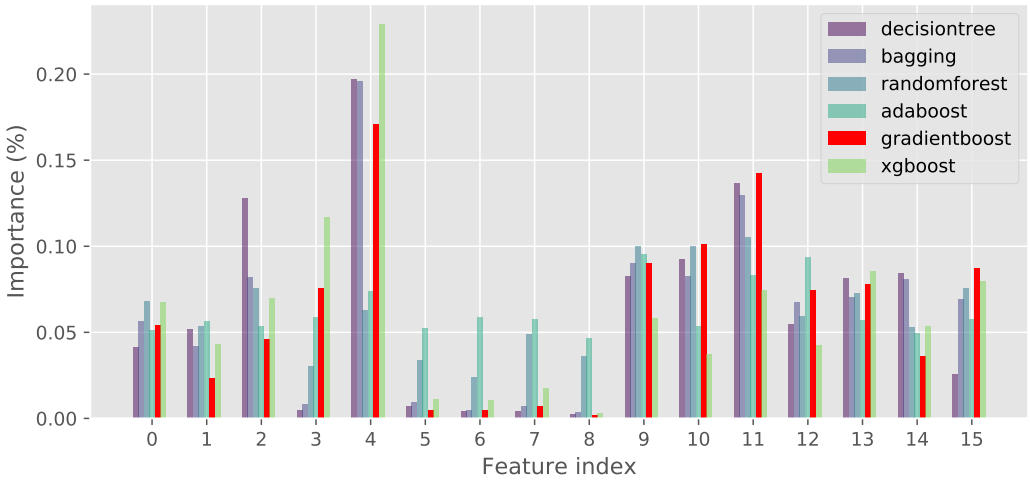

FIGURE 3.3. Feature importance. The feature indeces are found in table 2.1.

prediction accuracy is considerably lower for all the models, with a simple decision tree being the worst with 7.2% accuracy, and the gradient boosted trees the best with 19.5%. Since we have used data from the same subjects for training and validation, it is a challenging task to tune the hyper parameters of the models in this manner such that they generalize to the movement patterns of other subjects. A better approach to performing this task may have been to use data from separate subjects in training, validation and testing, as this likely would reduce the high variance for more biased models which generalize better.

Figure 3.3 gives an indication of what features are deemed as most important by the various algorithms. The ranges of all axes are in general not contributing much to the model, while features like the standard

deviation of the acceleration in y-direction and the velocity in y-direction are given high significance. These results might suggest that some features could be dropped from the dataset without worsening the results. However, because of the poor performance of the models in setting 2, it could be argued that adding additional features to the set is worth exploring; for example, some researches have had success with using high- and low-frequency filters on the accelerometer data[2]. Even so, the main findings of this analysis suggests that overfitting of the training data is primary problem.

Comparing our results with those of Ravi et al.[3] in table 3.2, we see that their decision tree, bagging and boosting classifiers achieve very similar accuracy scores as ours when mixing the subjects (setting 1), but there is a big difference in classification accuracy when separate subjects are used for training and testing (setting 2). It is not clear in their paper which hyper parameters they have used in their models, but we suspect that they are composed of trees with a considerably higher bias. They have also included correlation between the axes as predictors, but we chose to leave it out since our initial exploration of adding this feature seemed fruitless. Additionally, Ravi et al. used a different (though similar) dataset, so it is difficult to provide a direct comparison.

5. Conclusion

In this project we have studied the ability of tree-based and ensemble learning methods to recognize human activities based on accelerometer data. When using a training set containing data from all subjects (setting 1), both the single desicion tree, bagging, random forests and the three boosting methods gave a very high test accuracy score. The poorest performance was by the decision tree with 97.8%, and the best was by gradient boosting and XGBoost with 99.5%. Even though the methods can easily predict the target values when they are trained on all subjects, the overfitting problem becomes evident when testing the models on a set with separate subjects (setting 2). In this case the lowest score was 7.2% by the decision tree, and the highest was 19.5% by gradient boosting. These models were based on parameters tuned using a cross-validation grid search, which contributed greatly to the overfitting of the training data. Setting 2 is arguably the most realistic and interesting case, because it is much more useful to be able to use an activity recognition model on a subject without having to train the model specifically on data from said subject. The lack of generalization for models using setting 2 has also been experienced by other researchers[9] and seems to be a common challenge, because subjects perform the same activities in different ways. Our suggestion for future research is to experiment with using separate subjects for the training and validation set when performing cross-validation for parameter tuning, which might decrease the variance of the final result, and provide a more generalized model. We also explored some alternative ways of dealing with the overfitting problem, mainly by reducing the complexity of the models; this can be reviewed in the Appendix.

References

[1] Mitja Luštrek and Boštjan Kaluža. Fall detection and activity recognition with machine learning. *Informatica (Ljubljana)*, 33(2):205–212, 2009.

[2] Pierluigi Casale, Oriol Pujol, and Petia Radeva. Human activity recognition from accelerometer data using a wearable device. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6669 LNCS(October 2017):289–296, 2011.

[3] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data using symbolic data approach. *Lecture Notes in Networks and Systems*, 2005.

[4] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2009.

[6] Tianqi Chen. XGBoost : A Scalable Tree Boosting System.

[7] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. 2013.
[8] R.W. DeVaul and Steve Dunn. Real-time motion classification for wearable computing applications. *2001, Project Paper, http://www.media.mit.edu/wearables/mithril/realtime.pdf*, pages 1–14, 2001.
[9] Artur Jordao, Antonio C. Nazare, Jessica Sena, and William Robson Schwartz. Human Activity Recognition Based on Wearable Sensor Data: A Standardization of the State-of-the-Art. pages 1–11, 2018.

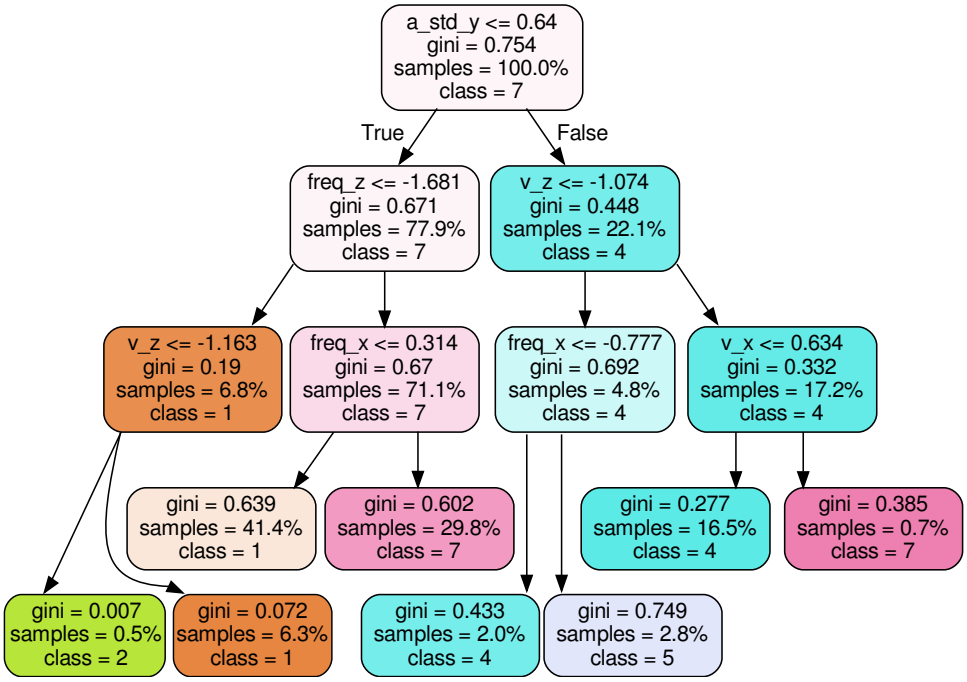

FIGURE 6.1. Decision tree for setting 2, and an accuracy of 40.3%.

6. APPENDIX

As mentioned in the discussion above, table 3.2 seemed to indicate that the models were highly overfitted to the training data, which resulted in terrible test performance in setting 2. In order to deal with this problem, we experimented with using trees with fewer splits. We achieved a significant increase in accuracy score by doing this, as shown in table 6.1. By reducing the tree depth to 3 for the decision tree, bagging and random forest, the test accuracies were raised to 40.3%, 40.2% and 45.1% for the respective methods. Figure 6.1 shows a graphical overview of the decision tree with a maximum depth of 3, and one thing to note is that only 5 of the 7 classes are represented by the leaf nodes, which is something that also should be considered when creating biased tree models with shallow depths.

The boosting methods were also improved by reducing both the max depth, the number of estimators and the learning rate, and XGBoost gave the best accuracy score, 60.5%, with a max depth of only 1. The resulting confusion matrix based on the XGBoost model is shown in figure 6.2. It is clear that the results have improved greatly for setting 2, but it is still far from usable in terms of separating the seven activities based on the dataset.

Another possibility is to simplify the targets, and check whether the learning algorithms perform better when dealing with activity categories, instead of the relatively specific activities we had in the original dataset. In order to investigate this, we combined the target in the following way:

- **1. Working at computer** (unchanged class 1).
- **2. Standing.** Combined class 3 ("standing") and 7 ("talking while standing").
- **3. Walking.** Combined class 4 ("walking") and 6 ("walking and talking with someone").

TABLE 6.1. Accuracy of classifiers on setting 2, when using a shallow depth in order to reduce overfitting.

Classifier	Estimators	Max depth	Learning rate	Accuracy
Decision tree	200	3	N/A	40.3
Bagging	200	3	N/A	40.2
Random forest	200	3	N/A	45.1
AdaBoost	50	1	0.01	35.3
Gradient boosting	50	1	0.01	43.6
XGBoost	50	1	0.01	60.5

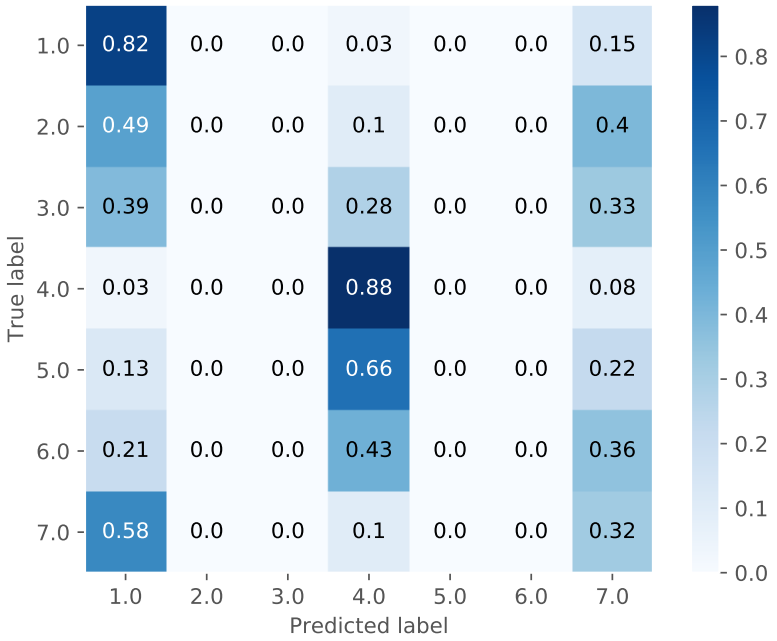

FIGURE 6.2. Confusion matrix using XGBoost, with parameters as specified in table 6.1, and an accuracy score of 60.5%.

- **4. Going up/down stairs** (unchanged class 5).
- Class 2 ("standing up, walking and going up/down stairs") was removed from the dataset, due to its complex nature.

With this simplified dataset, we created a new model using XGBoost, with 50 estimators, max depth of 3, and a learning rate of 0.01. The resulting accuracy score was 65.5%, and the confusion matrix is shown in figure 6.3. As we can see, this model is performing better, but still confuses some activity categories. The most notable confusion happens between class 3 and 4 of the simplified targets, "walking" and "going up/down stairs", which intuitively is reasonable, since these are similar activities that both involve walking.

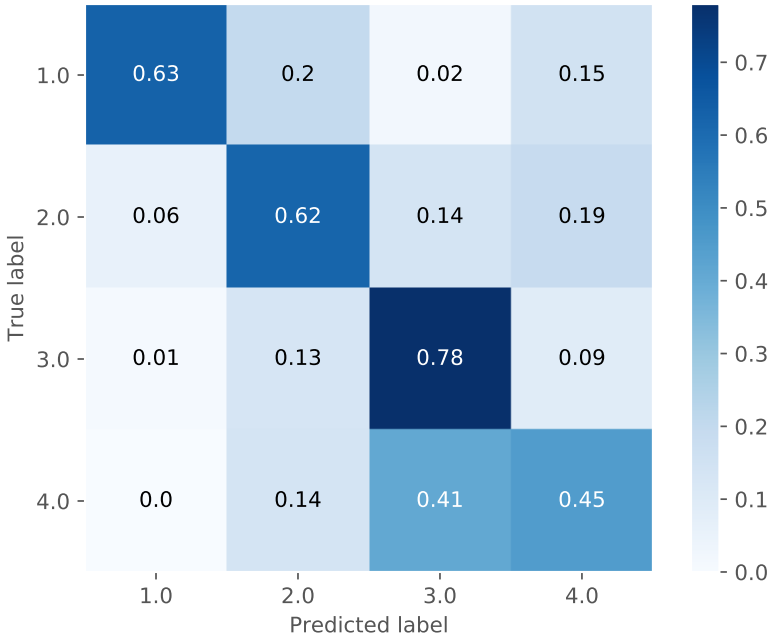

FIGURE 6.3. Confusion matrix using XGBoost on a simplified dataset, giving an accuracy score of 65.5%. Number of estimators was 50, max depth 3, and the learning rate 0.01.