Hack&Slash: Gesture Recognition for Dashboards

Seminar Thesis

Nico Rausch
Richard Tran
Giorgio Groß
1856484 - 1835121 - 1938927

At the Department of Economics and Management
Institute of Information Systems and Marketing (IISM)
Information & Market Engineering

Advisor: Erik Pescara (TECO)

20th of July 2017

Contents

1. Introduction

Nowadays gesture recognition has achieved a high usage and meaning, especially for medical purposes and industrial production processes. These new technologies are available not only for industrial purposes, but also for the mass market, for which devices like the Xbox Kinect or the Wii Remote set the basis a few years ago. Until now, many research groups (Baudel and Beaudouin-Lafon (1993), Ren, Yuan, and Zhang (2011), Rautaray and Agrawal (2015)) have already worked on technologies which enable the user to control computers with gestures. Most of these devices work vision-based, i.e. with cameras which recognize the visual changes caused by body movement and translate them into computer commands. These applications have the disadvantage that the user cannot move freely in the room, since he is bounded to the space captured by the camera in order to transmit the commands properly. This can be impractical for specific situations, e.g. presentations, when flexibility is needed considering the movement in the room.

Furthermore, there are also several technologies which track the physical data of the movement with the help of sensors (accelerometer, gyroscope, etc.) to recognize a gesture. These can be used independently from the location of the user in the room, but often demands expensive equipment, for instance a data glove. In order to develop a cheap and accurate sensor based gesture recognition device, different approaches use data from an accelerometer to classify gestures. These show a high accuracy, but are limited in the number of different identifiable gestures due to the less data.

In our work, we used the “Thunderboard React”, which is a development platform with several sensors including an accelerometer and a gyroscope. We examined, how well such a sensor platform is suited for gesture recognition and how it enhances dashboard user experiences. We did this by calculating “features” like the maximal acceleration from the collected data and classified these with different machine learning approaches. Overall, we established to recognize 4 different gestures with an expectancy accuracy of 95%. Furthermore, we created a dashboard to test the usability of our implementation.

2. Related Work

Gesture recognition with wearable devices has been an important research topic over the years. There have been a lot of works which used accelerometer-based devices, such as a Wii remote or a data glove. Xu, Zhou and Ji experimented with classifying seven hand gestures using the sign sequence of the gesture acceleration as the main feature (Xu, Zhou, and Li (2012)). They tried three different recognition algorithms in order to find the most suitable one. The movement data was gathered using MEMS 3-axis accelerometer with a Bluetooth connection to a PC. With a dataset of overall 628 samples they achieved an average recognition accuracy of 95.6% and they concluded, that in this experiment the recognition algorithm based on sign sequence and template matching was the most accurate algorithm. Liu also used template matching in order to classify accelerometer data (Liu, Zhong, Wickramasuriya, and Vasudevan (2009)). In their experiments, they utilized the three-axis accelerometer from the Wii remote as input device for their classifier. They evaluated a large gesture library with over 4000 samples for eight gesture patterns and their results show an overall 98.6% accuracy and a recognition delay of only 300 ms. Kim, Thang and Kim used several accelerometers, with which they created a data glove that was connected via Bluetooth with the PC (Kim, Thang, and Kim (2009)). They performed simple hand gesture recognition by considering a 10-point moving average of the acceleration with which they could achieve a recognition accuracy of 100% for only three distinct hand gestures.

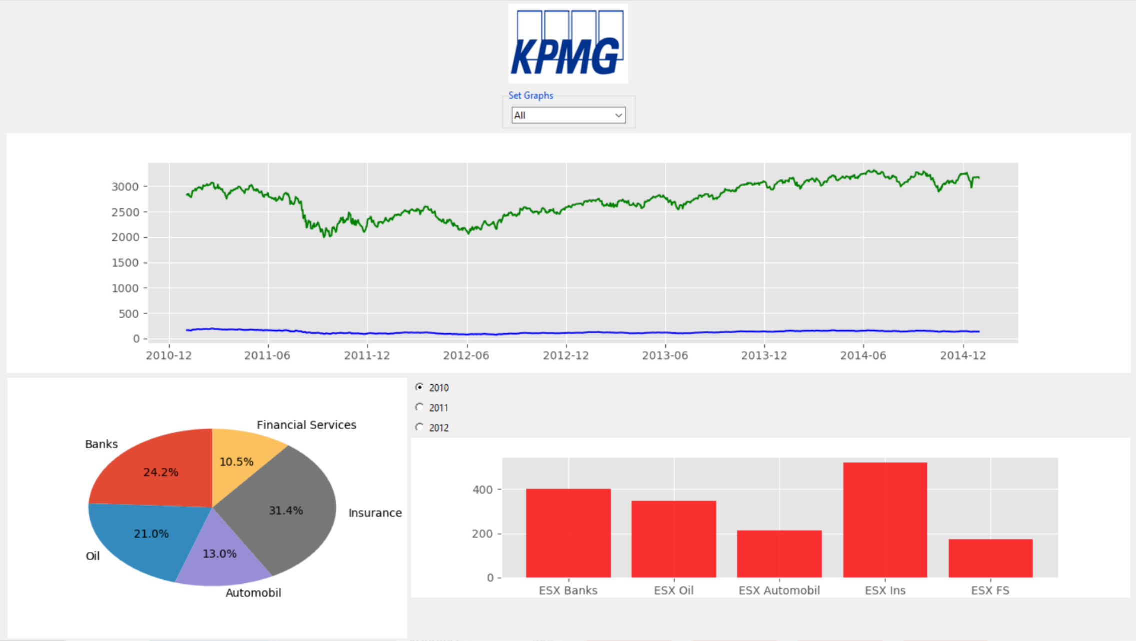
Further work has been done focusing on which machine learning algorithm performs best regarding gesture recognition. Belgioioso examined acceleration data collected from an iPod Touch and a HTC-Explorer smartphone. They experimented based on a dataset of overall 550 samples for four types of gestures and tested classification approaches such as Support Vector Machine (SVM), Relevance Vector Machine (RVM), Sparse Bayesian Learning (SBL) or k-Nearest Neighbor (kNN) (Belgiogioso, Cenedese, Cirillo, Fraccaroli, and Susto (2014)). For feature extraction, they employed a Principal Component Analysis which is an approach to reduce the dimensionality of a data set consisting of many variables correlated with each other. The work concluded that SVM has proved to be the approach with the highest accuracy in recognizing gestures, while SBL based approaches showed comparable classification performances with less computational cost required. To the best of our knowledge, this work presents the first gesture recognition study using the Thunderboard react.

3. Dashboard

The main goal of our seminar work was to implement a gesture recognition program with which we can reliably control applications like dashboards on the computer. Therefor, we created a test dashboard with which we examined the usability of our program. It consists of several diagrams which can be modified by steering elements like a dropdown menu and radio buttons. The main characteristic of the dashboard is that it can be controlled only with a few simple buttons. This makes the steering with our gesture recognition program easier since we can use the python package “pyautogui” to translate recognized gestures into commands on our user interface.

We decided to use the swipe lift and right gestures to select a value in a dropdown menu and we use a punch gesture to cycle through the available years in data.

Of course, this is just a simplified example of a dashboard with only a few features compared to what can be found in actual corporate dashboards which allow more sophisticated commands. Nevertheless, our dashboard can be easily extended with further features by implementing more steering elements like slide bars or check buttons and by adding more recognizable gestures with which these elements can be controlled.



Set Graphs: All

● 2010
○ 2011
○ 2012

Figure 3.1: Our demo application front end

4. System Design

In order to maximize reusability of code we decided to split the app into the six main components shown below. As you can see, our project consists of two individual parts, the recorder app and the recognizer app, sharing two helper modules, labeled Input and Connector. We implemented a core module for each part which is responsible to coordinate data streams and event responses. As such, the core modules instantiate their helper modules and use their interfaces to communicate with them. It is also worth knowing that some modules are divided into multiple files which are documented directly in our code.

4.1 Components

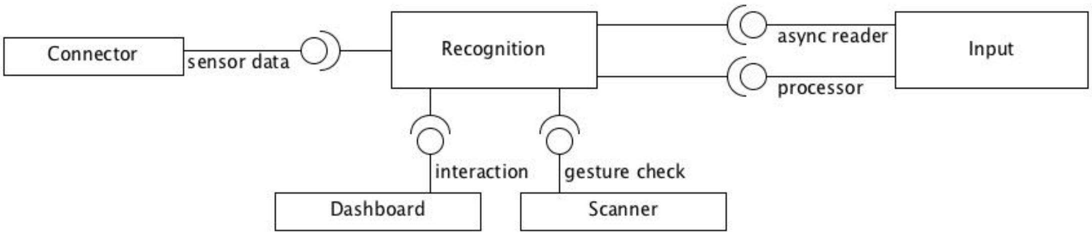


Figure 4.1: Recognizer App

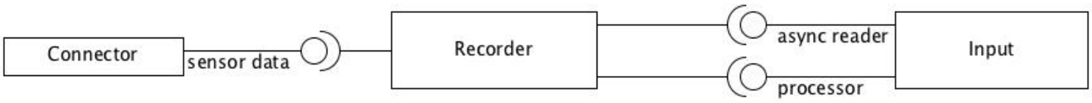


Figure 4.2: Recoreder App

4.1.1 Connector

As we ran into several issues trying to connect the Thunderboard React using Python we implemented the connector in JavaScript and used NodeJS libraries for interfacing the bluetooth device. To launch the recorder or recognizer app you need to launch the connector and pass it the path to the desired core module. After launching the connector it performs the steps shown in the flow chart on the right.

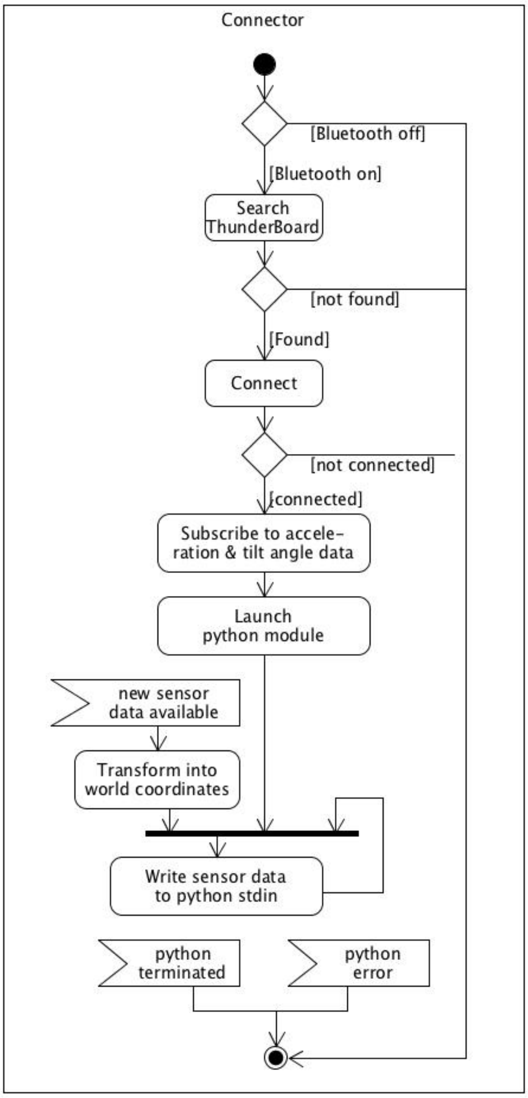


Figure 4.3: Establishing the connection to the Thunderboard React

4.1.2 Input

The input module starts a new thread upon instantiation to avoid blocking the calling thread. In its new thread it keeps reading the sensor data from stdin, parses it into floating point values and inserts them into a ring buffer. Its interfaces can be used to set up the buffer, to reset the buffer, to insert new data into the buffer or to start recording. In the latter case, the module finds out when the actual gesture started on its own by comparing the acceleration vector length to a threshold value. As soon as the buffer fills up its content is written to a static CSV file autonomously. The input module never takes action on its own but is still smart enough to perform a full control flow to record a gesture or to maintain the ring buffer.

4.1.3 Recorder

The recorder module provides a simple user interface which is used to specify the gesture you want to record as well as to explicitly start recording. That is done through a button which has to be pressed before each single gesture; we implemented it this way in order to avoid accidentally record a gesture which would pollute our training set. It also is responsible for instantiation and hooking up the input module.

4.1.4 Scanner

The scanner module is only used by the recognition module. It is used for monitoring the ring buffer content and for checking whether a gesture was observed. Thus, it instantiates, trains and maintains the machine learning algorithm used to classify live sensor data. Training includes transforming the recorded raw acceleration and tilt angle data into features and feeding them to the machine learning algorithm. All live sensor data will also have to be transformed into the same features so that they can be classified. We describe what features are and how we transformed the raw data in the next chapter. To avoid unnecessary complexity, we included the cross validation used to evaluate the performance of different machine learning algorithms directly into the scanner module. Cross validation includes leave one out, 10-fold cross validation and 25-fold cross validation and is performed during instantiation.

4.1.5 Dashboard

Just like the scanner, the dashboard module is only used by the recognizer app. It provides the user with a user interface showing charts and diagrams based on data by KPMG. That data is read and parsed by the dashboard when it is instantiated. As the user interface provides interactive elements like buttons the dashboard module is also responsible for handling user interaction. In addition, it exposes an interface to the recognition module which is used to broadcast recognized gestures as events to the dashboard. The dashboard can then execute the appropriate action.

4.1.6 Recognition

The recognition module used in the recognizer app is responsible to instantiate and hooking up the input module. It also instantiates the dashboard and implements the logic required to notify it about a recognized gesture.

5. Data Collection & Manipulation

Working with machine learning algorithms requires certain actions in order to choose, prepare and classify data streams. Choosing the right data is the key decision in order to be able to train the algorithm properly. In addition to this step, preparing data and transforming it into “features” is equally important.

5.1 Data Collection

To define which data will be collected we had to think about which kind of gestures we will want to recognize. Attaching the sensors to the back of our hand enables us to observe acceleration data for x, y, and z axis as well as tilt angles alpha, beta and gamma. We thought about attaching a magnet to our thumb so that we can use the hall-sensor as well, but we decided to leave this for further research. The collected acceleration and tilt angle data can later be used to distinguish gestures from each other. For collecting all the data we need, we use our recorder system, which reads data from the ThunderBoard by using our connector module and giving us the possibility to tag the recorded gesture with an ID. The ThunderBoard is therefor attached to the back of the right hand. To avoid recording and scanning data observed during normal gesticulation we require the acceleration vector length of each gesture to exceed a certain threshold. In our code we set this treshold to 1.2 G. As soon as the treshold is exceeded, our recorder saves the next four data points and the specified gesture ID as row in a CSV file. All in all, our team recorded 120 data sets per gesture, so 480 data sets overall. Because the recorded data is really slow to progress we decided to stop at this point, although we initially wanted external people to record some data for us.

5.2 Gestures

Really early in development, we thought about what gestures can we detect and what gestures are useful for us. We came up with many ideas, but the first two gestures we implemented were the two most intuitive gestures for us: Swipe Left and Swipe Right. For easy recognition of the gesture, we decided to put rules on the execution of these gestures. A „swipe left“ gesture consists of a very fast movement of the active hand to the left, followed by a slow movement back to the initial position. Both movements should not include a big tilt of the board. The „swipe right“ gesture is just the same, but with switched directions. By adding those rules we increased the success of recognition hugely. As a consequence, we removed the first data sets we recorded, without those rules, from the CSV. After a few weeks we reached a great success rate on the left and right swipes, so we added additional swipes „Up“ and „Down“ and a Punch gesture. The punch gesture is executed like a normal punch with the fist. Since the „Swipe Down“ and the Punch were often confound, we decided to exclude the „Swipe Down“-Gesture from our work.

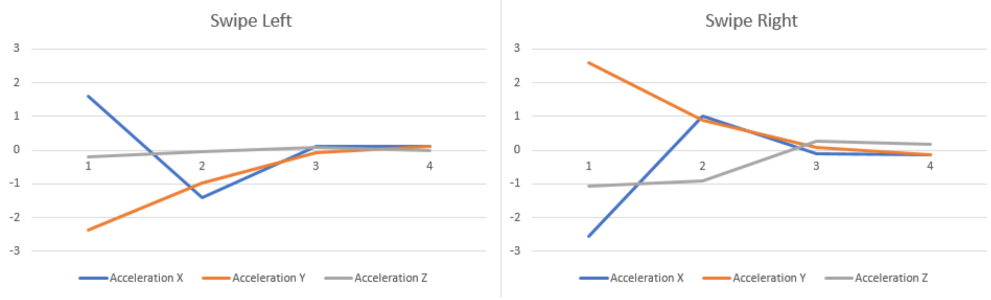


Figure 5.1: Choosing our features by looking at exemplary data points

5.3 Feature Design

Before we can train our algorithm, we have to transform the recorded data into a set of features. Choosing and designing those features affects the performance of the gesture recognition algorithm heavily.

5.3.1 Preprocessing

Standardization, scaling, normalization and binarization are common preprocessing techniques. Though, as the acceleration vector and tilt angle sizes are key characteristics to distinguish gestures scaling, normalization and binarization are not applicable for our use case.

5.3.2 Features

We first thought about using the variance as a feature, but early tests already revealed that the acceleration data often has the same variance across all of our tested gestures. We needed to find more useful features, that have different values when we execute different gestures.

The way we thought about this set of features was very intuitive. First, we recorded some gestures that we wanted to implement. We then created graphs for each of our data sets and looked what is characteristic for them. In Figure 5.1 for example, we could see that the Swipe Left (Swipe Right) gesture first has a minimum (maximum) on the X-Axis, then a maximum (minimum). So we decided to use maximum and minimum acceleration per axis, because they differ widely across gestures and are worth adding to our feature set.

To give our classifier a idea of which minimum/maximum was the first, we put them in the right order, so if we have a maximum first, this will be the first feature and the miniumum will become the second feature. For instance, swiping left first shows a negative minimum acceleration, e.g. -1.7G, and then a positive maximum acceleration, e.g. 1.2G, on the x axis. We now pick those values out of our buffer and figure out which one was earlier, so we can put them in our feature vector in the right order. In this example, we end up with the tuple (-1.7, 1.2).

To include the tilt angles into our feature set, we had to come up with a different approach. Like for acceleration, the variance of the given data wasn't a good idea either for angles, but also their minimum and maximum values are often equally great across different gestures. Anyway, taking the minimum and maximum into our feature set increase our precision a little bit, so we continued to use them.

Summing up all angles led to non-satisfying results, because they mostly sum up to a value near zero, so we then thought about more features per angle: One sums up the absolute values of negative angles and the other one sums up the absolute values of the positive angles.

Because the Thunderboard does a permanent recalibration, the absolute angles were not really useful. So, in the next step we decided to use only the differences between two angles in our buffer. This new approach improved our gesture recognition by about 5%, as you can see in picture 5.2.

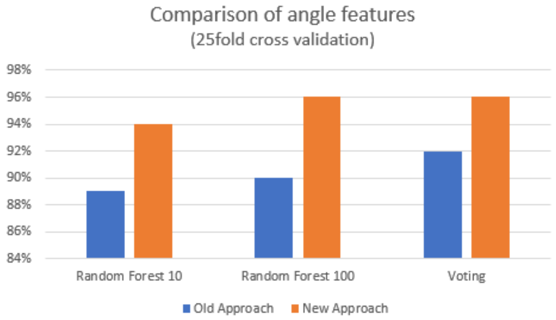


Figure 5.2: Comparison of our old and our new idea

5.4 Data Classification

Depending on the underlying training set, each classifier performs a bit different. We tested the K-Neighbors classifier with five and eleven nearest neighbors, Random Forest classifier with 10 and 100 estimators, Decision Tree classifier, Support Vector Machine and Gaussian Bayes, which is a neural network. We then created a Voting classifier based on the two best classifiers to push the limits even further. Each classifier was evaluated with leave one out and 25-fold cross validation techniques, the results are summarized in the diagrams below:

As you can see, the Gaussian Bayes and Support Vector Machine perform poorly compared to the other classifiers. Both their expectancy value and their standard deviation are unfavourable for practial use. In contrast, the Decision Tree classifier reaches almost 90% precision, accuracy and recall expectancy value. The prediction is based on a tree; inputs always iterate through each node until they reach a leave. Each node checks and classifies the input values and has one incoming and two outgoing edges. The leaves finally state which gesture the input data corresponds to. Using a tree is apparently a propmising solution, but as we end up in a leave the prediction has always a probability of 100% which limits this classifiers abilities to some extent. Combining multiple trees by using a so called Random Forest led to much better results. As shown in the diagram, the

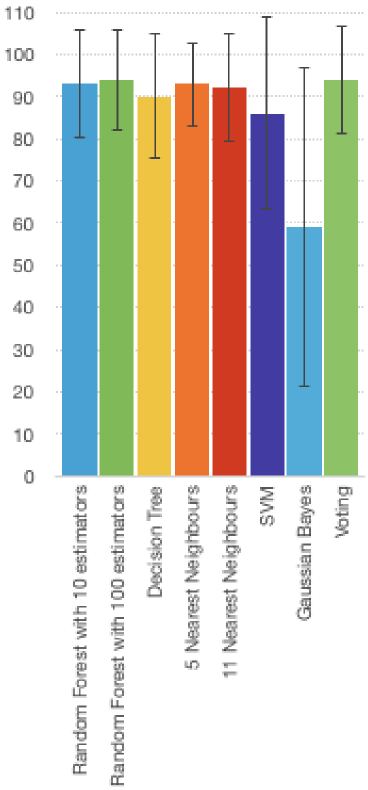


Figure 5.3: 25-fold cross validation precision

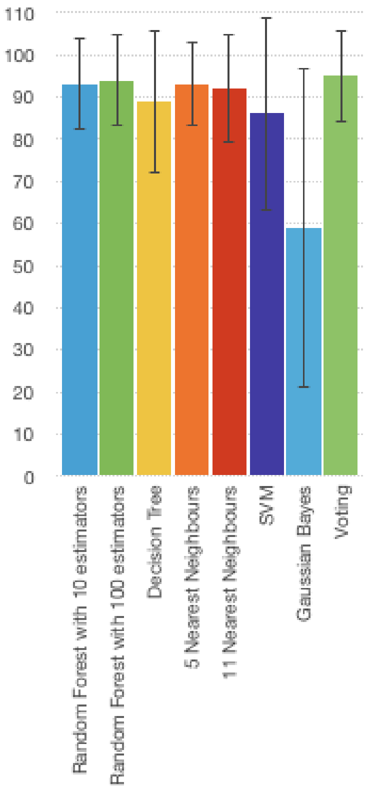


Figure 5.4: 25-fold cross validation accuracy

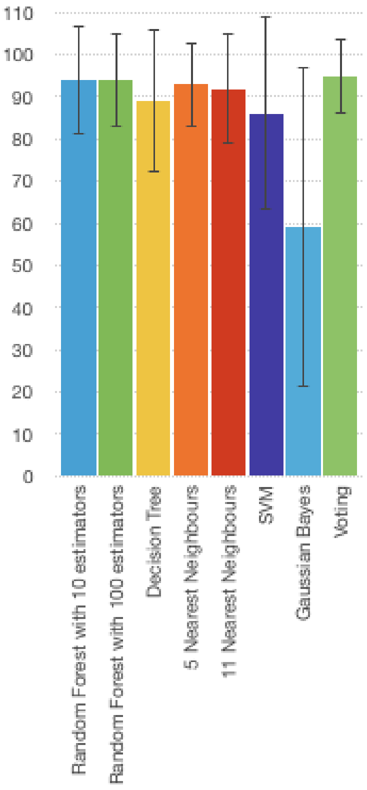


Figure 5.5: 25-fold cross validation recall

expectancy value of our classifier with 100 estimators reaches 94% precision, accuracy and recall.

The K-Neighbors classifier follows a differnet approach. Simply put, training data is mapped into a coordinate system. Input data is then mapped the same way and classified by checking the K nearest training data points in that coordinate system relative to the input data. Whatever gesture the majority of these K neighbors is assignet to is the predicted gesture of our input. In our case K was five or eleven. Considering just 5 neighbors showed off to be the better solution and compares very vell to our Desicion Tree classifier with 100 estimators: The K-neigbors classifier performs about 1% worse, but has a much smaller standard deviation equal to 10% in precision, accuracy and recall instead of 11% - 13%.

Having examined those common classifiers we combined the two best of them, Random Forest with 100 estimators and 5-Neighbors, into a Voting classifier. Tests revealed that giving the Random Forest predictions twice the weight of the 5-Neigbors predictions increases the expectancy value of our Voting classifier to 95% accuracy and recall while slightly decreasing the standard deviation. Due to these observations we decided to use the Voting classifier to control the dashboard.

In order to provide a convenient user experience gestures need to be predicted fast enough during app usage. We meassured the average prediction time for each classifier during our leave one out cross validation. The results are presented in figure 5.6 and justify our decision to use the Voting classifier for production. Although it has the highest prediction time we think that fiftysix milliseconds are still appropriate for real world applications. What is also worth noting is that the 5-Neighbors classifier needs approximately one millisecond and still performs almost as well as the Random Forest classifier with 100 estimators. So the relatively large prediction time of the Voting classifier is mainly caused by the Random Forest classifier we applied to the classifier set.

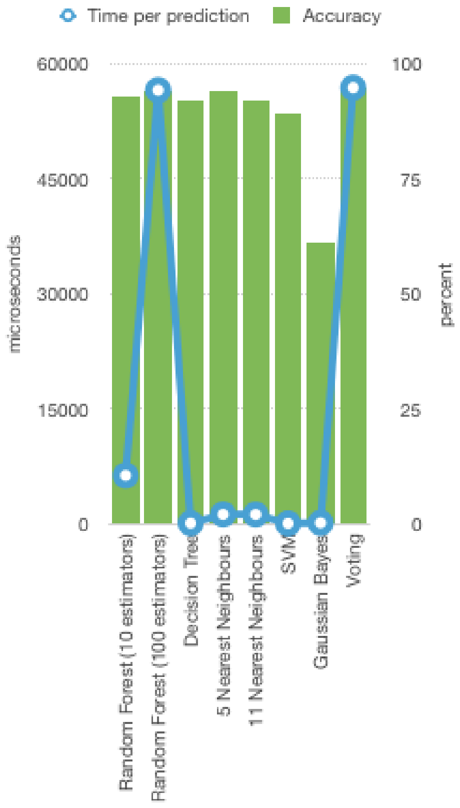


Figure 5.6: Leave one out cross validation accuracy

6. Testing

6.1 Functionality

In our tests, we noticed, that we can not fully reach the 95% of precision that we theoretically achieved in our calculations.

The key factor why we don't reach this precision is the very low frequency of data given to us by the ThunderBoard. We recieve only 5 data samples per seconds, our gestures are only half a second long. This leads to the problem, that one data point can mix , for example, the fast movement to the left and the slow movement back to the right, so we only get one datapoint from two different movements.

We tried to make this better by recording more data, in different speeds and with slightly different movement, but the more data we use, the algorithms heavily get slower.

In the end we are very happy that we now recognize more than 80% of the gestures and translate them into working dashboard commands.

ToDo

More?

ToDo

6.2 Usability

Usability tests with other people

7. Results

ToDo

Write a conclusion of our seminar

References

Baudel, T., & Beaudouin-Lafon, M. (1993). ”charade: remote control of objects using free-hand gestures”. *Communications of the ACM*, 28-35.

Belgiogioso, G., Cenedese, A., Cirillo, G., Fraccaroli, F., & Susto, G. (2014). ”a machine learning based approach for gesture recognition from inertial measurements”. *IEEE 53rd Annual Conference on Decision and Control*.

Kim, J., Thang, D., & Kim, T. (2009). ”3-d hand motion tracking and gesture recognition using a data glove”. *IEEE International Symposium on Industrial Electronics*, 1013-1018.

Liu, J., Zhong, L., Wickramasuriya, J., & Vasudevan, V. (2009). ”uwave: Accelerometer-based personalized gesture recognition and its applications”. *Pervasive and Mobile Computing*, 657-675.

Rautaray, S., & Agrawal, A. (2015). ”vision based hand gesture recognition for human computer interaction: a survey”. *Artificial Intelligence Review Volume 43*, 1-54.

Ren, Z., Yuan, J., & Zhang, Z. (2011). ”robust hand gesture recognition based on finger-earth mover’s distance with a commodity depth camera”. *Proceedings of the 19th ACM international conference on Multimedia*, 1093-1096.

Xu, R., Zhou, S., & Li, W. (2012). ”mems accelerometer based nonspecific-user hand gesture recognition”. *IEEE SENSORS JOURNAL*, 1166-1172.