Mobile Information Systems

Final Project Report

Topic: Research paper - Veritaps

Julian Steinhaußen: 122114

Ademola Eric Adewumi: 120769

‘Veritaps’ is a communication layer that can help app providers to identify truths and lies in mobile input. The final study proposes an accuracy(f-score) of .98 for truths and .57 for lies. Future application scenarios for Veritaps could be a security measure during online log-in, a trust layer during online sale negotiations, and a tool for exploring self-deception. (Mottelson, Knibbe, & Hornbæk, 2018) The input data processing, in this case, doesn’t refer to the actual content which i.e. is typed into a textbox but rather exploring a ‘content-agnostic’ approach across the whole range of inputs (sensor data etc.) to gather data about subtle behaviour.

The published study consists of three crowdsourced smartphone sub-studies with different measurement and testing methods for gaining knowledge about truths and lies in the context of mobile user input.

Study 1: Simple Lies

Task: Participants should tell truth/lie or choose randomly on the colour of the screen using 2 buttons or a slider

Measurements: Reaction/response Time

Results: On average, regardless of input type it took longer to tell a lie than the truth

Study 2: Ultimatum Game

Task: Share of fixed amount of money between 2 participants, the first participant proposes division of money, if both come to a match the money will be payed of, else there will be no pay-out

Measurements: Response time, Finger size, touch precision, button hold-down time, acceleration, rotation & signal magnitude, touch pressure

Results: lower mean acceleration(x,z) on truth entries (less hand movement), duration between 1st and 2nd key-event is higher for lies, classifier based on the obtained data

Study 3: Yatzi Game

· validation of classification results from study 2

Task: Yatzi with 12 rounds of rolls with 5 dices; participants were rewarded based on sum of reported scores (incentive to lie)

Measurements: same as in study 2 + swipe distance, duration, length, linearity and pressure of swiping and button press

Results: classification precision of 98% precision, and 97% recall for truths (F1 = .98), and 65% precision and 59% recall for lies (F1 = .57).

Finally, a feature selection out of these 3 studies was listed, to train a classifier for distinguishing between lies and truths. The selection consists of the following features:

• Num-pad: button precision (mean, min)

• Pressure: button pressure (mean, max, SD, pressure)

• Pressure: swipe pressure (mean)

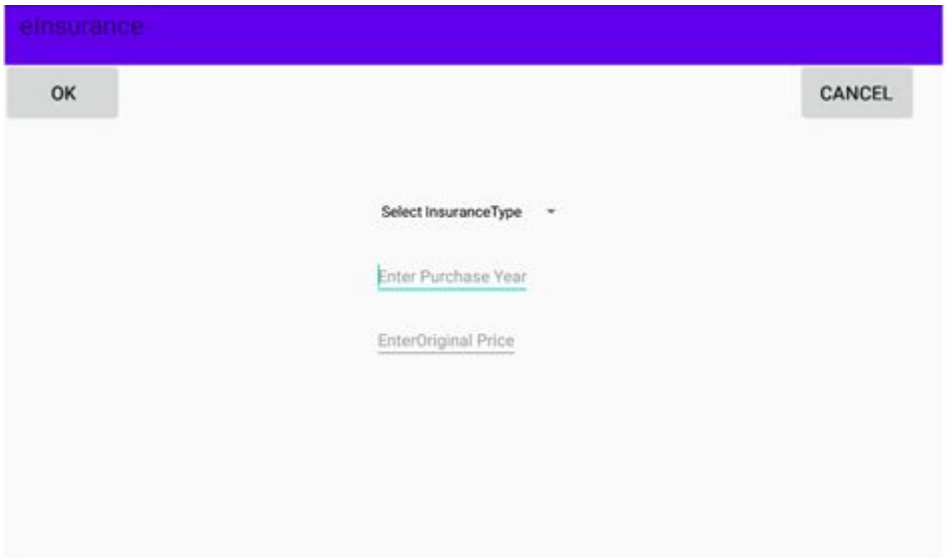
• Acceleration: x-acceleration (mean, max, SD)

• Acceleration: z-acceleration (SD)

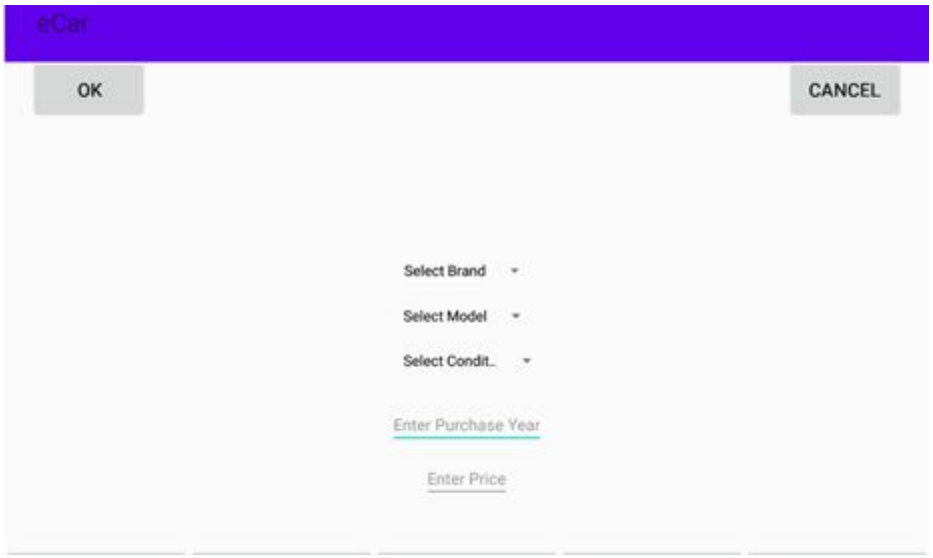
As mentioned in the beginning, the selection of features can, in addition to the classifier, help to identify lies in mobile input. A potential application which is listed in the study is a verification of an insurance claim. This potential application as well as the feature selection are the base for our mini project. While the implemented user interface is inspired by the provided example from the study, the feature selection is changed due to the lack of a device with a pressure sensor (devices[event.getPressure()] only return 0 for no touch event and 1 for touch event). Regarding this, the feature of touch pressure, in consultation with Prof Echtler, was swapped against response time.

Composition of the application

When the app is launched, the user sees the ‘eInsurance’ layout, where he can select the type of insurance he wants (through a spinner). Additionally, the user is intended to enter the purchase year and the original price into two edit text boxes. Through the ‘OK’ Button it is possible to navigate to the next layout. The ‘Cancel’ Button navigates back to the starting activity.



Based on the choice of insurance type (either eBike or eCar) the onClickListener of the ‘OK’ Button creates an Intent to the next activity which displays the next layout.

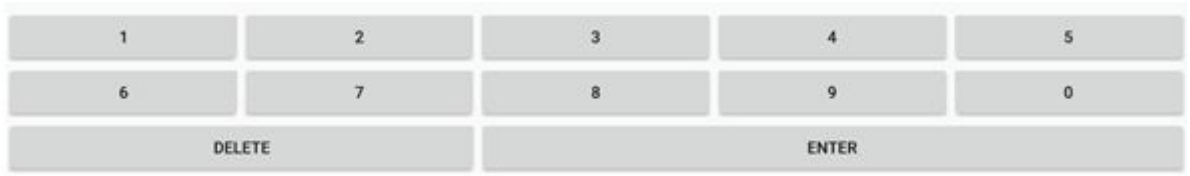


Again, there is a sample of spinners to provide different alternatives of vehicles to be insured. 2 edit text boxes take text input.

Both activities are listening to a broadcast receiver, triggered by the service class **‘AccelerationService’** which listens to value changes of the acceleration sensor of the device and broadcasts them to the receiver.

As stated, the application takes different time measurements. The time between the first and the text input is determined through a text changed listener ‘onTextChanged’ of the editable textboxes. Additionally, the time for the whole input process is measured.

The last part for measurement is the quantification of button/touch precision. For this, a custom keyboard was implemented, due to the inability of accessing the button locations of the provided soft keyboard from android.



The custom keyboard only provides a num-pad as well as a key for delete and enter and appears if the editable text boxes gain focus. The keyboard is implemented through a new input connection to the textboxes.

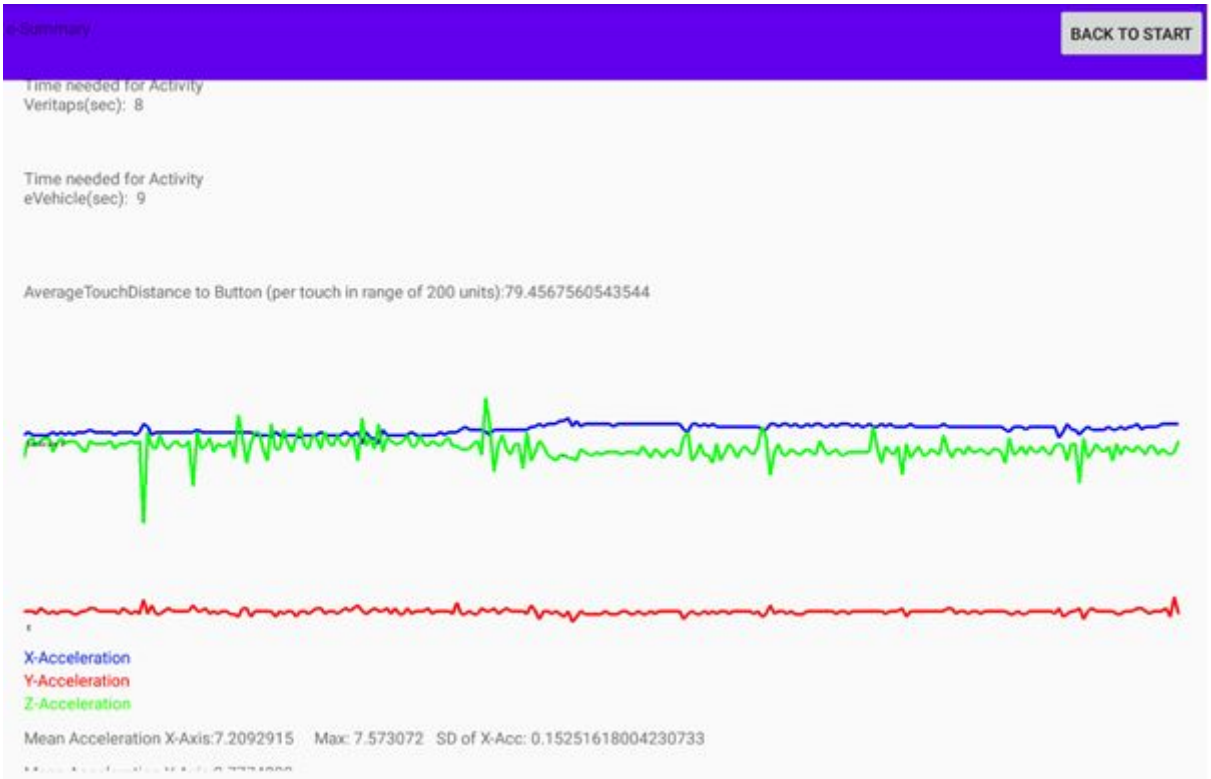
The button precision is measured with the help of the motion event **'dispatchTouchEvent'** in the activity before the touch event is consumed by the keyboard. The criteria for considering the touch events are:

- start of touch event (MotionEvent.Action_Down)
- distance to any key on custom keyboard within 200 units (to exclude accidently touches on the screen)

For each touch event the distance to the closest button is calculated, and if it is smaller then 200, it will be considered. Each activity creates an object of the class **'InsuranceData'** to store the gathered data in array lists. Before the activity is closed, the **save**() method is called, where the created object of 'InsuranceData' is saved in the sharedPreferences using the Json library to store the object as string.

If both, the UIs of eInsurance (called Veritaps in the app) and eVehicle(eCar or eBike in the app) have been filled out, the layout of the activity 'summary' is displayed. The summary class is responsible for processing the gathered data and displaying it to the user. The saved data in the sharedPreferences is decoded using Json again.

The acceleration data is displayed through a custom graph view, while the rest of the collected data is shown in textboxes. Both, the graph view and the textboxes are located inside a scroll view.



Advantages:

- User doesn't notice the gathering of user data during data collection
- The app works as specified
- Simplicity of user interface (user friendliness)

Shortcomings:

- Custom keyboard: user is familiar with android soft keyboard and won't use custom keyboard voluntary
- No real scale for the view of acceleration data
- Arrangement of layouts in the app is not nice (not uniform…)
- No classification of user input
- Arrangement of code inside the activities (readability suffers)
- Soft keyboard is not hiding on the virtual device (but hiding on real devices; tested: pixel 4(android 10), samsung galaxy tab s2 (android 7.0) and tecno ka70 (android 8.1.0))

Difficulties and what did not work well during the development:

- Trying to get the touch events from the soft keyboard from android (seemingly consumed by the view)
- Trying to locate the buttons of the soft keyboard
- Retrieving touch pressure from devices
- Which view/method should be used to intercept touch events
- Challenges hiding the soft keyboard on Virtual Device

Conclusion:

The developed app provides a basic structure for applications which want to evaluate the user input using the gathered content-agnostic input data. What is missing for the final usage is of course the classifier, which draws conclusions on the collected data. It consists of several activities where the user has to produce some inputs as well as an activity for summarising the results.

Problems during the development occurred mainly on the implementation of touch precision in cooperation with the android soft keyboard. Future work may focus on the implementation of the classifier, that, out of the provided basic structure, a real scenario application can be developed. For future projects, a concrete plan of implementations would be useful in terms of organization and readability of the code.

References

Mottelson, A., Knibbe, J., & Hornbæk, K. (2018). Veritaps: Truth Estimation from Mobile Interaction. CHI 2018 Paper.