Real-time Stress Assessment

1. Introduction and Background

Wearable devices are very common in our life nowadays such as Apple watch, and Samsung galaxy gear. The characteristic of wearable devices enables them to collect and exchange data between devices or platforms automatically. In terms of the application of wearable devices in health, there is still room for development. The advantage of wearable devices is that it can easily and automatically collect data, such as heart rate, body temperature, and physical information. However, current wearable devices such as Apple Watch are mainly used for information. They don't analyze or make conclusions based on those data. Today, there is a growing interest to use wearables not only for individual self-tracking but also within corporate health and wellness programs[1].

2. Problem Description and Challenges

The goal of our project is to develop a real-time stress detection system in general. We break down our problem into three parts. Firstly, we use the history of health data to build a stress assessment model. Then, we simulate the real-time sensor data from the data set and use the model to detect the stress level. Lastly, we display the result for information and further analysis. There are two challenges our project facing. One is to obtain an accurate model to detect the stress level. The other is to set an appropriate time interval to send data to ensure we can process the data in time.

3. Technical Details

3.1 Overview

The flowchart in Figure 1 describes how our project is built up.

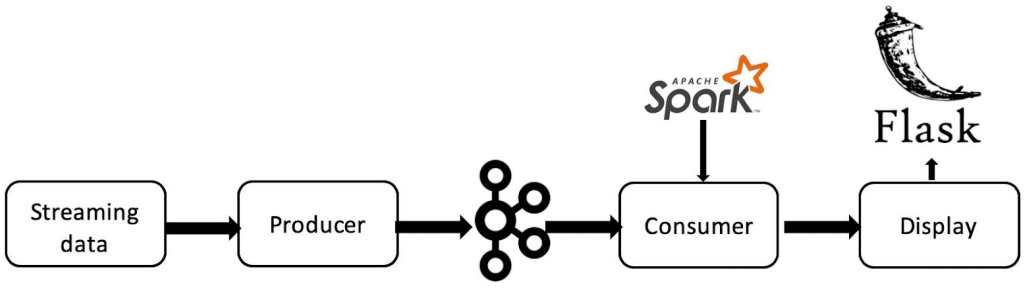

Figure 1. Project Flowchart

The first step is data preprocessing. We process the data and split the data into two data set before feed them into the model. One for training and the other one for testing. The second step is building the model, we train our data in Spark, using the random forest. In the third step, we build a streaming system. There are three python files in the system, *Streaming Data*, *Producer*, and *Consumer*. The *Streaming Data* file reads from the

processed data set and sends data to the producer. The *Producer* file sends data to Kafka at the same rate of every 5 seconds. The *Consumer* file extracts data from Kafka and uses the model to detect the stress level of users. The last step is to display the result on the web page for information.

3.2 Data Description

The dataset is obtained from UCI, *Wearable Stress and Affect Detection* [2]. It contains 15 subjects (people), and each person's data size is around 1GB. The total size of our raw data is 17.84GB. The label of the data (ground-truth) is from self-report of the subjects.

Figure 2 shows the raw data, it is generated at 700 *Hz* and contains 10 columns. The first one is the sequence number, the second column is label and the remaining columns are raw data from the sensor. We can see the detailed names of them in the form later.

0	0	27307	24015	32983	28535	37371	32903	31590	28355
1	0	27365	24013	32560	28517	37365	32900	31562	28363
2	0	27445	24006	33107	28527	37378	32903	31546	28371
3	0	27519	24010	33308	28527	37375	32916	31541	28374
4	0	27580	24012	32975	28535	37391	32916	31519	28382
5	0	27615	24010	32557	28521	37393	32924	31531	28383
6	0	27685	24013	31810	28532	37390	32933	31541	28396
7	0	27751	24012	32321	28527	37377	32942	31553	28397
8	0	27832	24015	32353	28537	37377	32953	31559	28409

Figure 2. Raw Data

Because the raw data cannot be used to train our model directly, we need to do some preprocessing to make our features trainable. There are two main steps in our preprocessing. The first one deletes the label with '0'. There are many '0' rows in our dataset, so the deletion number is large which is around 20% of the total data.

The second step is to convert the raw sensor values into SI units, the formulates are offered by the readme file in the dataset and we only need to write some mathematical function to deal with that. Table 1 shows our processing.

ECG (mV)	((signal/chan_bit-0.5)*vcc)
EDA (μS)	(((signal/chan_bit)*vcc)/0.12)
EMG (mV)	((signal/chan_bit-0.5)*vcc)
TEMP (°C)	vout = (signal*vcc)/(chan_bit-1.)rntc = ((10^4)*vout)/(vcc-vout)- 273.15 + 1./(1.12764514*(10^(-3)) + 2.34282709*(10^(-4))*log(rntc) + 8.77303013*(10^(-8))*(log(rntc)^3)
XYZ (g)	(signal-Cmin)/(Cmax-Cmin)*2-1, where Cmin = 28000 and Cmax = 38000
RESPIRATION (%)	(signal / chan_bit - 0.5) * 100

Table. 1 Raw Data Processing Formula

The following picture shows the data after preprocessing.

```
label,ACC_x,ACC_y,ACC_z,ECG,EMG,Temp,Resp
0,1.0785999298095703,–0.10460001230239868,0.5770000219345093,–0.0505828857421875,0.00091552734375,33.68225,–0.9857177734375
0,1.1242001056671143,–0.09579998254776001,0.5557999610900879,–0.0568084716796875,–0.0333709716796875,33.704956,–0.9979248046875
0,1.1510000228881836,–0.09780001640319824,0.4706000089645386,–0.0476531982421875,–0.06536865234375,33.70044,–0.970458984375
0,1.1602001190185547,–0.11119997501373291,0.3725999593734741,–0.0390472412109375,–0.0597381591796875,33.68979,–0.970458984375
0,1.1517999172210693,–0.1313999891281128,0.24500000476837158,–0.0403289794921875,0.0030670166015625,33.656464,–0.98419189453125
0,1.1317999362945557,–0.14340001344680786,0.09360003471374512,–0.0504913330078125,0.0167083740234375,33.704956,–0.97808837890625
0,1.1082000732421875,–0.14880001544952393,–0.0658000111579895,–0.0780487060546875,–0.0136871337890625,33.808136,–0.9857177734375
```

Figure 3. Preprocessed Data

Because the dataset is so large that if we use all of them to train the model, it exceeds the limitation of our memory and crash our computer. So we choose half of the samples from each person. After preprocessing and a random selection, our training data size is 4.78 GB and our test and demo data size is 3.9 GB.

3.3 Machine Learning Model

3.3.1 Data imbalance

By probing the dataset, we found that there is a disproportionate ratio of observations in each class, as shown in Table. An imbalanced dataset is bad for model generalization, and different models are sensitive to such imbalance in different degree, which makes it harder for us to select the actual good model according to the performance in the future.

label	number of entries
1	7374501
2	4105500
3	2338700

Table 2. Data Distribution over True Label

Our solution is to do downsampling on the data of the other two classes, which makes all three labels have 2338700 entries.

3.3.2 Training & Selection

An approximate test result on the dataset is shown in Table 3. They are all implemented in sci-kit learn.

Decision Trees		Random Forest		AdaBoost		KNN	
F1 -score	Acc	F1 -score	Acc	F1 -score	Acc	F1 -score	Acc
~78	~81	~90	~91	~89	~91	~64	~69

Table 3. Approximate Classification Result

Analyzing the result, we can see that random forest and AdaBoost are two models that are suitable for this task. Since the random forest multiclass classification algorithm is already implemented in Spark Machine Learning Library and easier to use for our system, we choose random forest as our model/

3.3.3 Model Fine-Tuning

After further tuning, the final parameters are shown in Table 4.

Parameter	Value
Number of Trees	100
Max Depth	9
Max Bins	32

Impurity Metric	Gini Index

Table 4. Model Parameters

3.3.4 Model Evaluation

The detailed performance of the model described in 3.3.3 is shown in Table 5.

	Class 1	Class 2	Class 3	General
precision	93.37	94.15	93.82	93.79
recall	93.05	90.23	97.95	93.97
F1	93.21	92.15	95.84	93.79

Table 5. Model Performances

3.4 Streaming data, Producer, and Consumer

3.4.1 Streaming data

When we test to send data from Producer to Kafka and then using Consumer to extract data from Kafka, there is a delay around 3 seconds. So, to make sure the data we send can be processed by the Consumer in time, we set the time interval we send data at a rate of every 5 seconds. The sensor generates data at a rate of 700 Hertz. Therefore, we send 3,500 rows of data every 5 seconds. Those data are written into a CSV file for the Producer to read.

3.4.2 Producer

In the Kafka Producer, we first construct a producer instance. We use confluent-Kafka, it shares the same set of configuration properties with librdkafka. The only required property needed is to specify the address of the broker in the Kafka cluster[3]. After configuring the parameter, we begin to send 3,500 rows of data to Kafka.

3.4.3 Consumer

The consumer part includes several steps to predict the received data and prepare the results for the visualization part. First, the Consumer receives 3500 rows per 5 seconds from the producer. The type of the data is string so we write a small function to transmit it into dataframe which can be used in the spark ML. Then we use a ready-made model to make a prediction by spark ML random forest multiclass classifier. Next, we use mode to merge 3500 input into one output. With this sampling strategy, we reduce the volume of the input data by retaining only an appropriate subset for analysis. This has a benefit is we can reduce the influence of error prediction to a very low level because our model accuracy is more than 0.95 so that even there are several wrong predictions, it can not be the mode. What's more, we also pack other parameters like temperature and ECG from 700 rows per second to 1 so that they can be updated in the visualization part every second.

3.5 Display

As to the final display, we use python and flask to make the frame and to the front end, we use echarts and d3.js to show our final results. The following left picture shows our front end, which is temperature updated per second, ECG updated per second and label updated every 5 seconds. The right picture shows the data made in consumer and used in visualization back end.

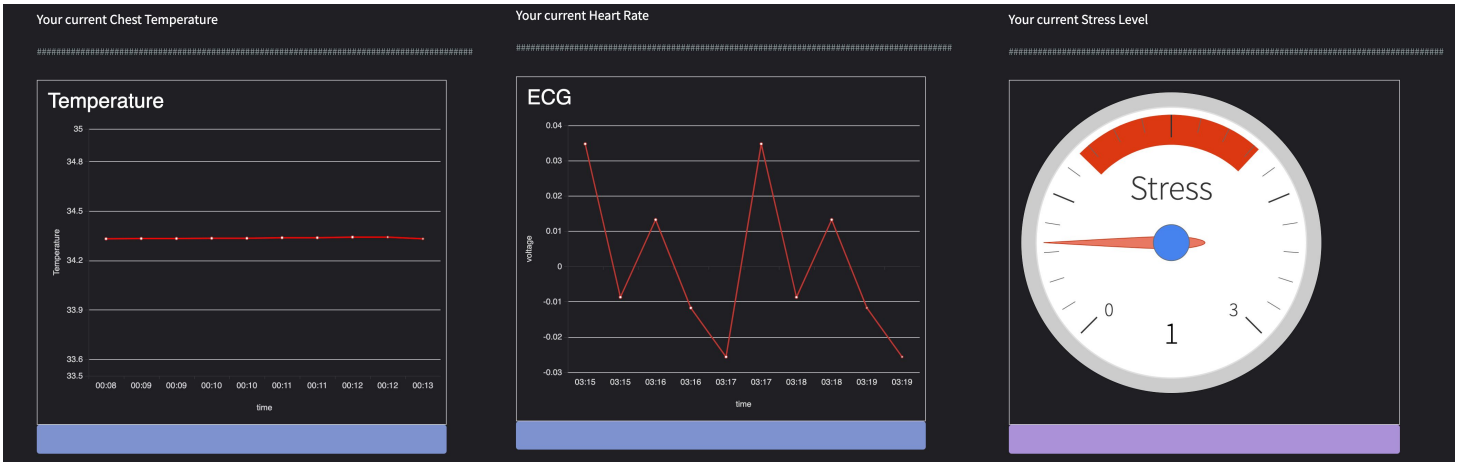

Figure 4. Data Visualization

4. Summary and Discussion

4.1 Throughput, latency, and scalability

All the simulations are run on one laptop. When there are only one producer and one consumer, the latency is around three seconds. When there are one producer and two consumers on the same laptop, the latency begins to increase. Consumers cannot process the data in the five-second interval. To solve the scalability problem, we need to separate the producer and consumer onto difference machines.

4.2 Limitation and future work

The system can only update every 5 seconds for the latency between the producer and the consumer. As a result, the seen data is from 5 seconds before. If we would like to have a system with less than 1s latency, we need to realize faster data transmitting. In addition, we can do some more detailed stress analysis (e.g. different stress level, integration with other types of medical report). For future work, we hope the system could integrate with could computing to accommodate real-time detection and handle multiple users at the same time. The right figure shows our plan for the future framework.

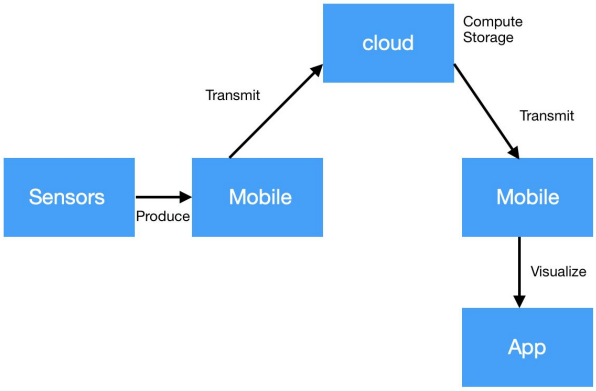

Figure 5. Future System Framework

References

[1] https://en.wikipedia.org/wiki/Wearable_technology

[2] Howlett, M. (2019, January 01). Introduction to Apache Kafka® for Python Programmers. Retrieved from https://www.confluent.io/blog/introduction-to-apache-kafka-for-python-programmers/ Müller, A. C., & Guido, S. (2017). Introduction to machine learning with Python: A guide for data scientists. O'Reilly.

[3] https://www.confluent.io/blog/introduction-to-apache-kafka-for-python-programmers/

[4] Schmidt, P., Reiss, A., Duerichen, R., Marberger, C., & Laerhoven, K. V. (2018). Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection. Proceedings of the 2018 on International Conference on Multimodal Interaction - ICMI '18. doi:10.1145/3242969.3242985