ISSN 2355-3286

Tahapan Desain dan Implementasi Model *Machine Learning* untuk Sistem Tertanam

Aminuddin Rizal
Teknik Komputer, Universitas Komputer Indonesia, Bandung, Indonesia
Email: aminuddin.rizal@umn.ac.id

Diterima 12 Oktober 2020
Disetujui 11 November 2020

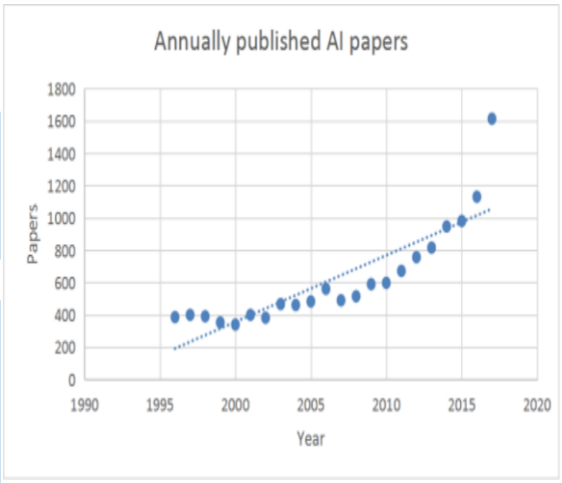
***Abstract*—Machine learning and edge computing currently becomes popular technology used in any discipline. Flexibility and adapt to the problem are the main advantages of its technology. In this paper, we explain step-by-step way to make a lightweight machine learning model especially intended for embedded system application. We use open source machine learning tool called as Weka to design the model. Moreover, we performed a simple stress recognition experiment to make our own dataset for evaluation. We evaluate algorithm complexity and accuracy for different well-known classifier such as support vector machine, simple logistic and hoeffding tree.**

Index Terms—edge computing, embedded system, machine learning, weka

I. PENDAHULUAN

An intelligence system atau sebuah sisem cerdas merupakan idaman bagi manusia di mana sistem tersebut dapat menganalisa permasalahan yang ada (*perception*) dan memberikan *feedback* secara natural (*decision making*). Penggunaan sistem cerdas saat ini meliputi banyak sektor seperti transportasi [1], ekonomi [2], pelayanan umum [3], pendidikan [4] hingga dunia medis [5].

Secara teknis untuk mewujudkan sebuah sistem cerdas beberapa metode dapat diimplementasiskan. Saat ini terdapat 2 teknik yang popular untuk digunakan yakni *machine learning* (ML) dan *deep learning* (DL). Gambar 1 [6] memperlihatkan peningkatan jumlah publikasi ilmiah yang membahas kedua metode tersebut, hal ini menunjukan kredibilitas metode yang disebutkan. Perbedaan mendasar dari *machine learning* dan *deep learning* ialah cara ekstraksi fitur. ML mengandalkan sebuah tahapan *preprocessing* untuk mendapatkan fitur yang diinginkan selanjutnya kita proses pada sebuah model. Sedangkan DL merupakan metode yang bersifat "*featureless extraction*" di mana fitur diproses di dalam sebuah model. Pada dasarnya, teori fundamental yang digunakan pada kedua metode tersebut meliputi probabilitas, statistika, kalkulus, dan aljabar linear. Sehingga sebuah model tidak lain hanyalah beberapa persamaan matematis yang dapat direalisasikan. Kedua metode tersebut memiliki keunggulan dan kelemahan masing-masing [7] dan secara khusus untuk ML yang memiliki kelebihan yakni ukuran modelnya yang relatif lebih kecil dan tidak kompleks.



Gambar 1. Tren jumlah publikasi ilmiah *machine learning* dan *deep learning* [6]

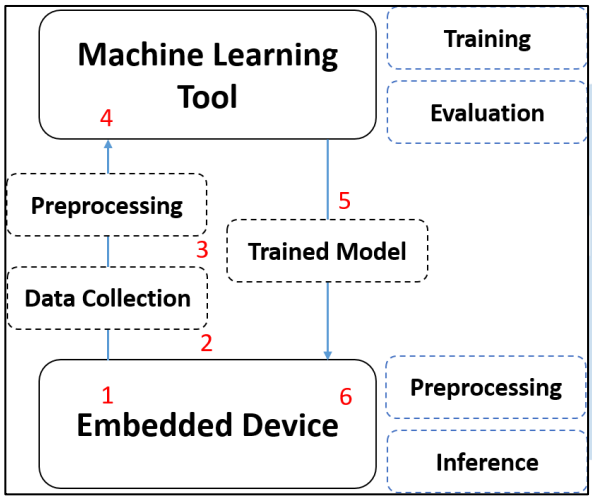
Penggunaan sistem cerdas idealnya dapat dilakukan secara *real time*. Biasanya untuk sistem yang kompleks dan menggunakan model yang berukuran besar spesifikasi komponen komputasi harus mengikuti. Sedangkan, tren saat ini telah dikenal dan sering digunakan sebuah teknologi yang disebut *edge computing*, di mana semua proses komputasi dilakukan di tempat sehingga menyediakan informasi dengan cepat. Hanya saja, komponen pemroses yang digunakan terbatas pada beberapa parameter spesifikasi. Sehingga saat menggunakan sebuah model sistem cerdas sebaiknya dengan kompleksitas dan ukuran yang kecil. Padahal, implementasi sistem cerdas pada *edge computing* memiliki prospek aplikasi yang luas salah satunya digunakan untuk penerjemah Bahasa isyarat [8]

Berdasarkan urgensi masalah di atas dan karena juga sedikitnya literatur mengenai implementasi ML untuk *edge computing* dalam Bahasa Indonesia, pada tulisan ilmiah ini kami menjabarkan langkah-langkah dalam mendesain dan mengimplementasikan algortima ML untuk apliaksi *edge computing* dalam hal ini kami menggunakan sistem tertanam. *Tools* yang kami gunakan pada penelitian ini bersifat *open source*

sehingga dapat digunakan oleh semua pihak. Terlebih, kami berikan contoh sederhana implementasi ML model pada *stress recognition* berdasarkan langkah-langkah yang kami jabarkan. Sebagai evaluasi, kami akan menguji kompleksitas dari model ML ketika ditananmkan pada sistem embedded yang digunakan.

II. Machine Learning Pada Embedded System

Secara sederhana proses desain dan implementasi algoritma *machine learning* (ML) pada sistem *embedded* dapat diilustrasikan pada Gambar 2. Pada dasarnya, 2 perangkat keras dibutuhkan untuk merealisasikan sistem ini, yakni *embedded device* (sistem yang akan kita implementasikan) dan *host computer* (digunakan untuk menjalankan *machine learning tool* untuk training dan evaluasi model).



Gambar 2. Implementasi model *machine learning* untuk sistem tertanam

Workflow dari implementasi model ML, secara berurutan ialah secara berikut,

1. Desain perangkat keras (*hardware*) dari sistem tertanam sesuai dengan tujuan dan spesifikasi yang dibutuhkan
2. Kumpulkan data dengan baik dan benar dan merepresentasikan penggunaan sebenarnya
3. Mendapatkan fitur dari data yang terkumpul (*features extraction*) dan memberi label atau *class* pada kumpulan data tersebut (*labelling*)
4. *Train* model pada *host computer* dengan menggunakan ML *tool*, pilih *classifier* yang cocok digunakan
5. Test model yang didapat dengan beberapa metrik pengujian (misal *K-fold validation*)
6. Lakukan optimasi dengan memilih fitur mana yang dibutuhkan dan memiliki korelasi dengan permasalahan yang dihadapi
7. Implementasikan model yang sudah dilatih ke dalam sistem *embedded* yang telah dibuat
8. Dalam penggunaan secara *real time* data *stream* harus diproses terlebih dahulu untuk mendapatkan fitur yang sama fitur yang digunakan untuk *training* model
9. Terakhir untuk memprediksi *class* yang sesuai, proses *inference* dapat dilakukan

Berikut detail dari Gambar 2 mengenai cara implementasi model ML ke dalam sistem tertanam.

A. *Embedded Devices*

Saat ini komponen komputasi yang digunakan pada sistem tertanam memiliki banyak pilihan, dari berbagai macam tipe mikrokontroller hingga beberapa tipe mikrokomputer yang popular digunakan di Indonesia antara lain ialah,

1. Arduino berbasis ATmega328 – RISC 8 bit mikrokontroller berjalan pada kecepatan 16 MHz
2. ESP32 – 32 bit Xtensa LX6 SoC dengan *clock source* 160 MHz
3. ARM seri M – RISC 32 bit mikrokontroller dengan basis clock 72 MHz
4. ARM seri A – arm64 arch mikrokomputer yang berjalan dengan menggunakan sistem operasi (misal Raspberry Pi)

Dari beberapa contoh di atas, pertimbangan pemilihan komponen komputasi tergantung kepada permasalahan yang ingin di atasi, karena komponen tersebut memiliki kelebihan dan kekurangan masing-masing. Parameter yang dapat menjadi pertimbangan seperti interface I/O, kecepatan (*clock speed* semata bukan menjadi *benchmark*), kapasitas memori, tipe data yang dapat digunakan (penting untuk ML) dan konsumsi daya (penting untuk *edge computing*)

B. *Data Collection and Preprocessing*

Melatih sebuah model ML membutuhkan data yang cukup banyak untuk mendapatkan hasil yang lebih terpercaya. Oleh sebab itu pengumpulan data dengan baik dan benar perlu dilakukan. Data yang dikumpulan merupakan data dari sensor, contoh: suhu, kelembapan, *inertial motion unit*, elektrokardiograf, dan banyak lainnya. Pengumpulan data harus dilakukan sesuai dengan kriteria sinyal dan sistem yang diinginkan contoh yang menjadi konstrain adalah frekuensi *sampling* dan tipe data yang didapat (*signed, unsigned, fixed point, floating point*). Terlebih, data tersebut mewakili beberapa atribut dan *class* yang akan diprediksi.

Setelah data yang diinginkan terkumpul, proses selanjutnya yaitu memproses data tersebut untuk mendapatkan fitur ML yang baik. *Preprocessing* akan berbeda sesuai dengan spesifikasi dan kebutuhan masing-masing sinyal. Salah satu parameter yang perlu diperhatikan ialah panjang sebuah segmen atau *window* dari sebuah sinyal. Membuat fitur dari satu data *stream* tidaklah direkomendasi karena apabila sinyal tersebut sebuah noise akan membuat fitur yang kurang bagus. Dengan demikian biasanya sebuah segmen (kumpulan

dari beberapa n poin sinyal, *buffer*) sinyal dibuat. Ukuran segmen akan berpengaruh dengan respon dan akurasi prediksi. Semakin panjang segmen maka akan kurang responsif, namun memiliki banyak informasi di dalamnya. Segmen tidak hanya dalam bentuk *time domain*, namun bisa juga dalam bentuk *frequency domain*. Terlebih lagi bisa dalam bentuk dari gabungan kedua domain tersebut, seperti yang dilakukan pada karya ilmiah [9].

Selanjutnya salah satu proses yang paling penting ialah mengekstraksi sebuah segmen sinyal menjadi fitur ML. Fitur yang paling umum digunakan ialah statistika fitur seperti rata-rata, median, modus, maksimum, minimum, standar deviasi, kuartil dan banyak lainnya [10]. Fitur-fitur tersebut dikombinasikan agar mendapatkan keunikan yang lebih terlihat pada sebuah sinyal.

Setelah perhitungan fitur selesai, kita dapat memberi label pada fitur tersebut dan membuat sebuah database fitur beserta labelnya yang biasa disebut dataset. Format dataset bermacam, yang umum digunakan kolom merepresentasikan bebera fitur, dan juga label dari fitur tersebut.

C. *Model Training and Evaluation*

Dari dataset yang sudah dibuat, selanjutnya kita dapat melakukan proses *training* model dengan menggunakan ML tool pada *host* komputer kita, spesisfikasi *host* komputer kita berpengaruh terhadap lamnya proses training. Terdapat beberapa pilihan yang dapat digunakan dari yang gratis hingga berbayar, diantaranya yang popular ialah,

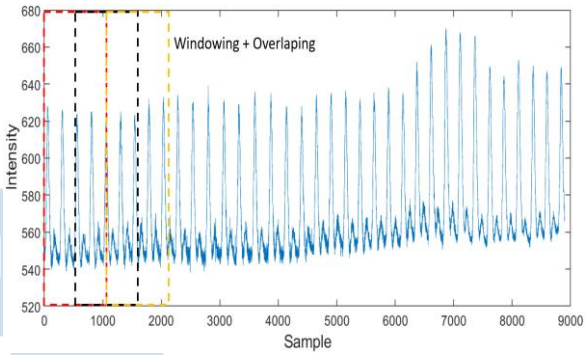
1. TensorFlow – *free*, berbasis python, *script interface*
2. PyTorch – *free*, berbasis python, *script interface*
3. Caffe – *free*, berbasis C++, *script interface*
4. Weka – *free*, berbasis Java, *GUI interface*
5. Rapid Miner – *paid*, berbasis Java, GUI *interface*
6. Matlab – *paid*, berbasis Java, GUI *interface*

ML dapat dibagi menjadi 2 kategori *learning*, yakni *supervised* dan *unsupervised learning*. Penggunaanya tergantung dari aplikasi yang diimplementasikan. Jenis kategori tersebut berpengaruh pada tipe *classifier* yang harus digunakan, dan juga konfigurasi parameter pada *classifier* tersebut juga harus diperhatikan.

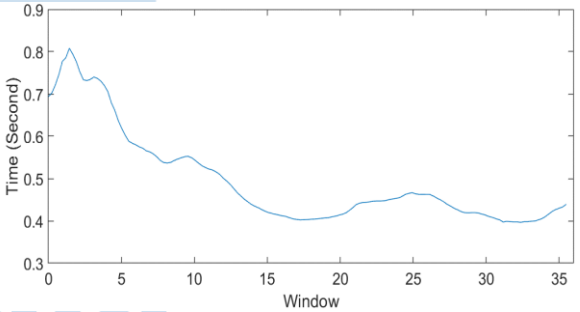
Setelah model kita *train*, selanjutnya kita dapat mengevaluasi kinerja dari model tersebut secara offline. Yakni dengan cara split data ataupun cara umum yang digunakan yakni *K-fold cross validation* (K adalah angka bulat menandakan pembagian data, misal 5, 10, 20). Hal ini perlu dilakukan untuk mengetahui apakah model yang dihasilkan memiliki performa yang bagus. Apabila tidak, maka kita dapat melakukan optimasi dengan cara memilih *classifier* yang lain, mengganti konfigurasi parameter pada *classifier* tersebut, ataupun dapat dilakukan dengan cara mencari fitur yang sesuai dan dibutuhkan untuk menyelesaikan masalah yang dihadapi.

D. *Model Implementation on Embedded Device*

Apabila model yang didapat sudah memiliki performa yang baik, maka model tersebut dapat kita implementasikan ke sistem tertanam kita. Cara implementasi dapat dilakukan dengan cara mencari persamaan atau model matematika yang terdapat di dalam model tersebut. Selanjutnya model tersebut dapat dijalankan secara *real time*. Sebelum melakukan proses *inference*, *raw* data harus di proses dengan proses yang sama saat membuat dataset.



Gambar 3. *Raw PPG signal*



Gambar 4. Fitur: *mean RR*

III. CONTOH IMPLEMENTASI

Pada contoh implementasi kali ini kami menggunakan sinyal photopletysmography (PPG) yang kami rekam untuk mengetahui tingkat stress yang dialami oleh pengguna. Implementasi menggunakan mouse PPG dan sebuah stress test game yang dapat dilihat detailnya pada referensi ilmiah [11]. Terdapat 3 kelas yang diukur yakni, "no stress", "time pressure", dan "interruption". Machine learning tool yang digunakan pada tahap evaluasi ialah Weka dengan bantuan Matlab untuk tahap pengolahan sinyal nya. Berikut detail dari masing-masing tahapan yang kami ilustrasikan seperti Gambar 2,

A. *Embedded Device*

Mikrokontroler yang digunakan yakni NUC120 berbasis ARM®Cortex™-M0. Mouse tersebut dilengkapi dengan Bluetooth modul dan 4 sensor PPG dengan *analog front end* (AFE) meliputi filter dan

programmable-gain amplifier (PGA117). Sensor menggunakan frekuensi sampling 250 Hz.

B. Data Collection and Preprocessing

Data diambil dari beberapa subjek, tetapi untuk penelitian ini kami hanya mengambil satu sebagai contoh. Subjek diminta untuk meletakkan telapak tangan di mouse (seperti menggunakan mouse) sembari melakukan aktivitas sesuai dengan aplikasi yang kami buat. Di dalam eksperimen kita dapatkan 4 sinyal PPG dari 4 sensor berbeda, tetapi untuk tujuan kali ini kami gunakan salah satu sensor yang memiliki *signal-to-noise ratio* (SNR) paling besar yakni pada sensor yang terletak pada jari.

Selanjutnya sinyal kami analisa dan kita proses untuk membuat fitur ML-nya. Pada tahap ini yang kita miliki ialah *raw signal* yang dapat dilihat pada Gambar 3. Pada sinyal 1 dimensi seperti sinyal PPG sinyal setiap fitur diambil dari sinyal yang sudah melalui proses *windowing* dan *overlapping*. Proses *windowing* yakni mengambil beberapa sample dari sinyal secara *real-time* untuk mendapatkan resolusi yang tinggi pada fitur-fitur yang dibuat. Panjang *window* berbeda antara karakteristik sinyal dan sistem yang digunakan. Kami menggunakan 3 detik window (750 *samples*) dengan panjang *overlapping* 1 detik (250 *samples*). Lalu kita dapat mengekstrak fitur dari setiap *window* sinyal. Untuk penelitian ini kami memberi contoh penggunaan 4 *statistical features* yang sering digunakan dalam *stress recognition* yakni "Mean RR", "RMSSD", "LF", dan "HF". Dengan mengaplikasikan operasi pada persamaan (1) sampai (4) secara berturut-turut. Di mana, PR ialah *pulse rate* yang merupakan nilai denyut jantung dalam satu menit, RR ialah *peak-to-peak interval* dari sinyal PPG yang memberikan informasi periode pada sinyal, RMSSD ialah *root-mean square* dari RR interval, LF ialah total dari *power spectrum* di frekuensi 0.04 sampai 0.15 Hz, HF ialah total *power spectrum* di frekuensi 0.15 sampai 0.4 Hz. Contoh gambaran fitur pada fungsi waktu dapat dilihat pada Gambar 4 yang merupakan nilai fitur dari "Mean RR".

$$RR(second) = \frac{60}{PR} \quad (1)$$

$$RMSSD = \sqrt{\frac{1}{r-1}\sum_{k=1}^{r-1}(RR_{k+1} - PR)^2} \quad (2)$$

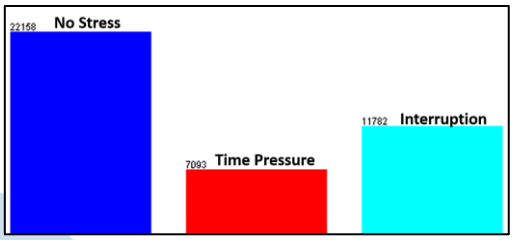
$$LF = \frac{\sum_{f1=0.04}^{0.15} P_{NN}(f1)}{\sum_{f2=0}^{fs/2} P_{NN}(f2)} .100 \quad (3)$$

$$HF = \frac{\sum_{f1=0.15}^{0.4} P_{NN}(f1)}{\sum_{f2=0}^{fs/2} P_{NN}(f2)} .100 \quad (4)$$

Fitur yang sudah dikalkulasi dan diekstrak lalu dibentuk menjadi sebuah dataset untuk dapat digunakan di dalam *machine learning tool*.

C. Model Training and Evaluation

Dataset yang dibuat bisa kita *import* ke dalam Weka. Sebaiknya dalam membuat dataset seharusnya dalam jumlah yang sama (*balanced*) sehingga dapat meningkatkan akurasi dari model yang akan dibuat. Sedangkan pada penelitian ini kami menggunakan *imbalanced dataset* dengan distribusi *instance* dapat dilihat pada Gambar 5, di mana kelas "No Stress" memiliki jumlah yang paling banyak diikuti dengan kelas "Interruption" dan kelas "Time Pressure". Tiga *classifier* kami gunakan sebagai contoh, yakni support vector machine, simple logistic, dan hoeffding tree, dengan konfigurasi masing-masing *classifier* sebagai berikut,



Gambar 5. Distribusi Dataset

- Hoeffding tree
  - *decimal number* = 2
  - *batch size* = 100
  - *grace period* = 200
  - *hoeffding tie threshold* = 0.05

  Setelah dilakukan proses *training* model yang dihasilkan terlihat seperti pada Gambar 6. Model hoeffding membentuk pohon logika.

```
LF <= 1521.861:
|   LF <= 227.334: no stress (414.500) NB1 NB adaptive1
|   LF > 227.334:
|   |   MEAN_RR <= 948.797:
|   |   |   MEAN_RR <= 587.016: time pressure (121.000) NB2 NB adaptive2
|   |   |   MEAN_RR > 587.016:
|   |   |   |   MEAN_RR <= 784.427: no stress (4619.401) NB3 NB adaptive3
|   |   |   |   MEAN_RR > 784.427:
|   |   |   |   |   MEAN_RR <= 799.741: no stress (474.500) NB4 NB adaptive4
|   |   |   |   |   MEAN_RR > 799.741: no stress (2045.500) NB5 NB adaptive5
|   |   MEAN_RR > 948.797: time pressure (2201.207) NB6 NB adaptive6
LF > 1521.861: interruption (3515.001) NB7 NB adaptive7
```

Gambar 6. Model Hoeffding Tree

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       22587               55.0459 %
Incorrectly Classified Instances     18446               44.9541 %
Kappa statistic                          0.0508
Mean absolute error                      0.3561
Root mean squared error                  0.456
Relative absolute error                 89.6178 %
Root relative squared error            102.291  %
Total Number of Instances            41033
```

Gambar 7. Hasil *Testing* Hoeffding

- Support vector machine (SVM):
  - *one-versus-one* SVM
  - *decimal number* = 2
  - *seed number = 1*
  - *batch size* = 100.

Setelah dilakukan proses *training* model yang dihasilkan terlihat seperti pada Gambar 8. Pada gambar terdapat 3 persamaan berbeda yakni (dari atas ke bawah) persamaan untuk "no stress" vs "time pressure"; "no stress" vs "interruption"; "time pressure" vs "interruption". Pada model ini semua persamaan dioperasikan dan dicari nilai maksimum sehingga didapatkan kelas yang diprediksi

```
        2.3807 * (normalized) MEAN_RR
+       2.5474 * (normalized) RMSSD
+      -3.7529 * (normalized) LF
+       0.0106 * (normalized) HF
-       2.1626

        0.002  * (normalized) MEAN_RR
+      -0.0042 * (normalized) RMSSD
+       0.0057 * (normalized) LF
+       0.0004 * (normalized) HF
-       1.0001

       -5.6278 * (normalized) MEAN_RR
+       2.3668 * (normalized) RMSSD
+       0.5107 * (normalized) LF
+      -2.6365 * (normalized) HF
+       2.1102
```

Gambar 8. Model SVM

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       25147               61.2848 %
Incorrectly Classified Instances     15886               38.7152 %
Kappa statistic                          0.2518
Mean absolute error                      0.3644
Root mean squared error                  0.4247
Relative absolute error                 91.7018 %
Root relative squared error             95.2832 %
Total Number of Instances            41033
```

Gambar 9. Hasil *Testing* SVM

- Simple Logistic
  - *batch size* = 100.
  - *heuristic stop = 50*
  - *max boosting iterations = 500*
  - *decimal number = 2*

Setelah dilakukan proses *training* model yang dihasilkan terlihat seperti pada Gambar 10. Pada model ini pula semua persamaan yang ada di model dioperasikan sehingga mendapatkan nilai maksimum yang berkorelasi dengan kelas yang diprediksi.

```
Class no stress :
3.74 +
[MEAN_RR] * -0 +
[RMSSD] * -0.01

Class time pressure :
-2.66 +
[MEAN_RR] * 0    +
[LF] * -0

Class interruption :
-0.75 +
[LF] * 0
```

Gambar 10. Model Simple Logistic

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       25297               61.6504 %
Incorrectly Classified Instances     15736               38.3496 %
Kappa statistic                          0.3094
Mean absolute error                      0.3213
Root mean squared error                  0.4137
Relative absolute error                 80.8592 %
Root relative squared error             92.8007 %
Total Number of Instances            41033
```

Gambar 11. Hasil *Testing* Simple Logistic

Setelah model di *training*, model tersebut dievaluasi sehingga kita dapat mengetahui performa dari model tersebut. Pada penelitian ini kami menggunakan 10-*Fold Cross Validation* untuk melakukan evaluasi model. Hasil dari masing-masing model terlihat pada Gambar 7, 9, dan 11 untuk model hoeffding tree, SVM, dan simple logistic secara berurutan. Rata-rata hasil yang didapatkan keseluruhan model menunjukan akurasi di bawah 70 %, yang berarti kurang baik untuk diimplementasikan. Peningkatan akurasi dari sebuah model dapat dilakukan dengan beberapa cara, yakni salah satunya dengan menyeleksi fitur mana yang berdampak besar pada performa prediksi dan dapat menghilangkan fitur yang sifatnya redundan. *Feature selection* dapat dilakukan di dalam weka, *rule-of-thumb* dari seleksi fitur ialah melihat korelasi fitur tersebut dengan fitur yang lain sehingga dapat mempersempit dimensi fitur pada model. Ilustrasi pemilihan fitur pada weka terlihat pada Gambar 12. Kami memberi contoh penggunaan *principle component analysis* (PCA) yang difungsikan sebagai penyeleksi fitur dengan mencari korelasi antara fiturnya. Setelah model dievaluasi dan berfungsi dengan baik. Maka bisa diimplementasikan ke dalam *embedded device*.

```
Correlation matrix
  1      0.33   0.08  -0.55
  0.33   1      0.89   0.33
  0.08   0.89   1      0.32
 -0.55   0.33   0.32   1


eigenvalue      proportion      cumulative
  2.08711         0.52178         0.52178       -0.664RMSSD-0.653LF-0.355HF-0.072MEAN_RR
  1.54486         0.38622         0.90799       -0.758MEAN_RR+0.621HF-0.19RMSSD-0.062LF
  0.33176         0.08294         0.99093       0.644HF+0.541MEAN_RR-0.528LF+0.116RMSSD

Eigenvectors
 V1       V2       V3
-0.072   -0.7579   0.5414 MEAN_RR
-0.6645  -0.1896   0.1164 RMSSD
-0.6534  -0.0617  -0.5282 LF
-0.3555   0.6212   0.6437 HF

Ranked attributes:
 0.47822   1 -0.664RMSSD-0.653LF-0.355HF-0.072MEAN_RR
 0.09201   2 -0.758MEAN_RR+0.621HF-0.19RMSSD-0.062LF
 0.00907   3 0.644HF+0.541MEAN_RR-0.528LF+0.116RMSSD
```

Gambar 12. *Feature Selection*

D. *Model Implementation*

Cara paling sederhana menanamkan model ke dalam *end device* dengan cara membuat program persamaan dari model tersebut. Sebagai contoh seperti Gambar 13 yang merupakan bentuk s*nippet* program (Bahasa C++) dari model simple logistic. Ataupun cara lain yakni dengan menggunakan API dari *machine learning tools* yang dipakai, karena pada contoh kali ini kami menggunakan weka, maka dapat menggunakan API yang weka miliki yang berbasis Java.

IV. DISKUSI

Dari tahapan yang kami jabarkan seperti Gambar 2, masing-masing proses memang sangat penting. Dari pemilihan jenis device yang digunakan hingga bagaimana cara mengimplementasikan model ke dalam sebuah program. Di dalam menggunakan *machine learning* memang tujuan utamanya ialah performa dari model tersebut bukan kompleksitasnya. Tetapi dalam *end device* kita memiliki limitasi pada sumber komputasinya. Pada contoh kali ini kami melakukan eksperimen dengan menggunakan 3 *classifier*, dan salah satu implementasi model terlihat pada Gambar 13 (simple logistic *classifier*). Dalam model tersebut dapat direpresentasikan menjadi 3 baris snippet program, yang mana sangat memungkinkan untuk dijalankan pada *end device* yang memiliki kemampuan komputasi yang rendah sekalipun. Sedangkan model yang kompleksitasnya agak tinggi (berbasis *tree*) dapat direpresentasikan menjadi sebuah program dengan menggunkan *syntax if-else* atau *case* bercabang. Kompleksitas model yang kami contohkan memiliki O(1) kompleksitas sehingga dapat mudah untuk dijalankan.

```
void predict (double mean_rr, double rmssd, double lf, double, hf)
{
    double *temp[3];
    int    *pos;

    temp[0]   = 3.4 + mean_rr * -0 + rmssf * -0.01;
    temp[1]   = -2.66 + mean_rr * 0 + lf * -0;
    temp[2]   = -0.75 + lf * 0;

    const int N = sizeof(temp) / sizeof(int);

    pos = distance(temp, max_element(temp, temp + N));
    return pos;

}
```

Gambar 13. *Embedding to device*

Hasil yang didapatkan akan berbeda setiap *resource* dan aplikasi yang dibuat. Contoh aplikasi-aplikasi yang memiliki *workload* yang besar seperti *natural language processing* (NLP). Maka prosesor yang digunakan untuk aplikasi tersebut memerlukan proses komputasi yang lebih tinggi contohnya menggunakan *single board PC* (SBC). Serta metode *machine learning* yang digunakan akan lebih kompleks dibanding dengan aplikasi yang sederhana.

V. SIMPULAN

Machine learning merupakan sebuah metode yang sangat popular untuk digunakan di dalam beberapa disiplin ilmu yang ada. Hanya saja terkadang kompleksitas model dari ML menjadi limitasi bagi beberapa *processing unit* dengan kemampuan yang rendah seperti untuk sistem tertanam. Kami memberikan contoh alternatif bagaimana cara membangun model ML yang dapat digunakan di perangkat komputasi rendah. Dari beberapa model yang kami buat kompleksitas model ML kami berada di kisaran O(1) untuk persamaan langsung dan O(n) untuk model *tree*.

DAFTAR PUSTAKA

[1] W. Xia, D. Zhou, Q. Xia and L. Zhang, "Design and implementation path of intelligent transportation information system based on artificial intelligence technology," in The Journal of Engineering, vol. 2020, no. 13, pp. 482-485, 7 2020.

[2] Y. Chen, W. Lin and J. Z. Wang, "A Dual-Attention-Based Stock Price Trend Prediction Model With Dual Features," in IEEE Access, vol. 7, pp. 148047-148058, 2019.

[3] O. S. Al-Mushayt, "Automating E-Government Services With Artificial Intelligence," in IEEE Access, vol. 7, pp. 146821-146829, 2019.

[4] L. Chen, P. Chen and Z. Lin, "Artificial Intelligence in Education: A Review," in IEEE Access, vol. 8, pp. 75264-75278, 2020.

[5] Y. Wei et al., "A Review of Algorithm & Hardware Design for AI-Based Biomedical Applications," in IEEE Transactions on Biomedical Circuits and Systems, vol. 14, no. 2, pp. 145-163, April 2020.

[6] L. Yang and M. Zhu, "Review on the Status and Development Trend of AI Industry," 2019 IEEE 4th International Conference

on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China.

[7] M. Sewak, S. K. Sahay and H. Rathore, "Comparison of Deep Learning and the Classical Machine Learning Algorithm for the Malware Detection," 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Busan, 2018, pp. 293-296

[8] B. G. Lee and S. M. Lee, "Smart Wearable Hand Device for Sign Language Interpretation System With Sensors Fusion," in IEEE Sensors Journal, vol. 18, no. 3, pp. 1224-1232.

[9] E. W. Sinuraya, A. Rizal, Y. A. A. Soetrisno and Denis, "Performance Improvement of Human Activity Recognition based on Ensemble Empirical Mode Decomposition (EEMD)," 2018 5th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Semarang, 2018, pp. 359-364.

[10] A. Kamal and M. Abulaish, "Statistical Features Identification for Sentiment Analysis Using Machine Learning Techniques," 2013 International Symposium on Computational and Business Intelligence, New Delhi, 2013, pp. 178-181.

[11] Lin, Shu-Tyng et al. “A Pulse Rate Detection Method for Mouse Application Based on Multi-PPG Sensors.” Sensors (Basel, Switzerland) vol. 17,7 1628. 14 Jul. 2017, doi:10.3390/s17071628.