Analisis Algoritma *Multiclass Image Classification* menggunakan *Decision Tree*

Rayhan Kinan Muhannad - 13520065[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13520065@std.stei.itb.ac.id

***Abstract*—*Image classification* merupakan sebuah fitur mesin yang memungkinan mesin tersebut mengenali dan mengklasifikasikan *image*. Fitur ini dapat dibuat dengan menggunakan metode *supervised machine learning*, yaitu memberikan mesin kumpulan data beserta label untuk "dipelajari" oleh model matematikanya. Terdapat beberapa jenis *supervised machine learning*, salah satunya adalah *decision tree*. Metode *decision tree* menggunakan struktur data *tree* sebagai model matematikanya dan partisi dengan paramater *information gain* terbesar sebagai algoritma pembuatannya. Agar *decision tree* dapat memproses data dalam bentuk *image*, maka diperlukan metode *feature extraction* dalam bentuk warna, bentuk, tekstur dari *image*. Implementasi model *decision tree* pada makalah ini direalisasikan menggunakan bahasa Python dengan bantuan modul sklearn, numpy, cv2, dan lain-lain.**

Keywords*—*Data Processing, Decision Tree, Image Recognition, Supervised Learning

I. PENDAHULUAN

Pada zaman modern ini, batas antara manusia dengan teknologi sudah semakin pudar. Sudah banyak teknologi yang dapat melakukan hal-hal yang sebelumnya dikira hanya manusia yang bisa melakukannya, seperti memproses ucapan manusia, mengidentifikasi gambar, mengerti bahasa natural manusia, merespons percakapan, belajar dari error yang dilakukan, dan masih banyak lagi. Konsekuensinya, akan semakin sulit membedakan antara mana yang manusia dengan mana yang mesin. Pada tahun 1950, Alan Turing sudah memprediksi bahwa hal tersebut akan terjadi, dimana interaksi manusia dengan mesin tidak bisa dibedakan dengan interaksi manusia dengan manusia. Beliau merancang suatu percobaan yang dinamakan Turing Test. Di dalam percobaannya itu, partisipan diminta untuk berinteraksi dengan beberapa orang dan/atau mesin. Interaksi tersebut dapat berupa *chat box* atau percakapan langsung. Kemudian, partisipan diminta untuk mengidentifikasi percakapan mana yang berlangsung dengan manusia dan percakapan mana yang berlangsung dengan mesin.

Selain memproses ucapan dan percakapan manusia, suatu mesin juga dapat mengidentifikasi suatu objek atau gambar layaknya manusia. Manufaktur mobil listrik terkenal, Tesla, Inc., mengintergrasikan fitur *image recognition* dan *decision making* dalam algoritma komputer mobilnya. Dengan teknologi itu, mobil Tesla dapat mengemudi automatis tanpa campur tangan dari pengemudi. Algoritma di dalam mesin *embedded* mobil Tesla dapat mengidentifikasi objek, memprosesnya, kemudian mengambil keputusan yang optimum dengan sangat cepat. Menurut suatu studi yang diadakan oleh National Highway and Transportation Safety Authority (NHTSA), 94% kecelakaan kendaraan terjadi akibat human error. Studi tersebut juga menambahkan bahwa *self-driving car* dengan *control degree* tertentu dapat membantu mereduksi persentase kecelakan kendaraan akibat *human error*.

Image recognition juga dapat digunakan di berbagai aspek dan bidang, seperti dalam bidang *security* dengan bentuk perangkat lunak *facial recognition* atau *retinal scan*; dalam bidang kedokteran dengan bentuk perangkat lunak pendeteksi jaringan tumor atau kanker; dalam bidang kebumian dengan bentuk perangkat lunak *topological terrain modelling*; dan masih banyak lagi.

Suatu mesin dapat dikatakan memiliki kemampuan *image recognition* apabila mesin tersebut memenuhi 3 kriteria dibawah ini:

1. Mesin dapat memproses data masukan dalam format *image* (png, jpg, jpeg, dll).
2. Mesin dapat memodelkan data masukan dalam bentuk model matematika dan/atau statistika.
3. Mesin dapat memproses data masukan baru dan mengidentifikasi *image* tersebut berdasarkan model yang dibentuk dari data masukan awal.

Seperti yang sudah dipaparkan di atas, suatu mesin perlu menerima data masukan awal dan membuat model yang bersesuaian dengan data itu. Data masukan awal yang diterima mesin harus ekstensif dan mencakupi keseluruhan fungsionalitas dari model yang diharapkan. Oleh karena itu, untuk memperoleh mesin yang *fine-tuned* diperlukan data yang relatif banyak. Selain volume data yang banyak, data yang dimasukkan juga haruslah bersih. Data yang kotor berpotensial mengubah model secara masif. Umumnya sebelum pemasukan data ke dalam model, dilakukan terlebih dahulu *data preparation and cleaning*.

Setelah dilakukannya *data preparation and cleaning*, pemrogram perlu terlebih dahulu menentukan jenis model yang cocok untuk jenis prediksi yang hendak dilakukan oleh model. Pemrogram perlu menganalisis kegunaan model tersebut, apakah model tersebut digunakan untuk melakukan prediksi secara kontinu (biasanya dalam bentuk regresi) atau untuk melakukan prediksi secara diskrit (biasanya dalam bentuk klasifikasi); atau model tersebut harus melakukan *unsupervised learning* (mesin dapat menganalisis data tanpa adanya target yang ditentukan oleh pemrogram, salah satu contohnya adalah algoritma *recommender system*) atau *supervised learning*

(mesin dapat menganalisis data hanya jika ada label atau nilai target yang ditentukan oleh pemrogram, salah satu contohnya adalah algoritma image recognition). Proses-proses yang telah dipaparkan tersebut dikenal sebagai proses *machine learning* atau proses pembelajaran mesin.

Dalam makalah ini, penulis akan lebih fokus terhadap pembahasan *machine learning* dengan jenis model *supervised learning*. Seperti yang sudah dipaparkan sebelumnya, model *supervised learning* memerlukan data yang memiliki suatu label atau nilai yang akan diprediksi oleh model. Sebagai contoh, misalkan terdapat seorang pemrogram yang hendak membuat model *machine learning* dengan tujuan untuk memprediksi suatu nilai saham pada pasar saham. Pemrogram tersebut pertama-tama harus mengumpulkan data-data nilai saham dari beberapa tahun ke belakang. Kemudian, pemrogram tersebut harus menyeleksi data mana yang terbilang bagus (tidak mengandung nilai defek) dan mencakup kebutuhan model untuk melakukan prediksi yang akurat. Selanjutnya, pemrogram harus memilih model yang tepat guna untuk memprediksi saham. Untuk kasus ini, penulis mengasumsikan pemrogram tersebut menggunakan model *logistic regression and classification*. Model tersebut diharapkan dapat memprediksi apakah pemegang saham harus menjual atau membeli suatu saham (memprediksi klasifikasi label diskrit) berdasarkan data-data yang dijadikan basis model.

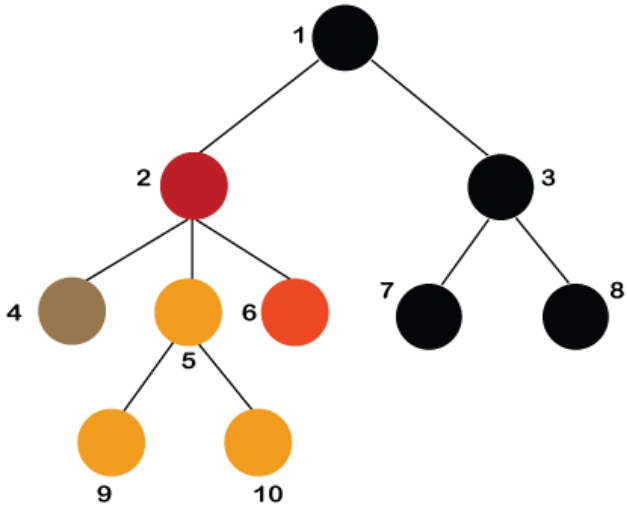
Terdapat banyak jenis model *supervised learning* selain *logistic regression and classification* yang digunakan sebagai contoh diatas, seperti *linear regression* and *classification*, *decision tree*, *random forest*, *neural network*, dan *support vector machine*. Setiap model tersebut memiliki kelebihan dan kekurangannya masing-masing. *Linear and logistic regression* memerlukan mesin dengan *computing power* yang rendah, tetapi hanya bisa memodelkan data yang relatif sederhana. *Decision tree* memiliki tingkat akurasi yang tinggi serta dapat diaplikasikan ke berbagai jenis data, tetapi cenderung terjadi *overfitting* ketika menangani data yang besar. *Random forest* memiliki tingkat akurasi yang lebih tinggi dari *decision tree*, tetapi memerlukan *computing power* dan memori yang besar. Mirip dengan *random forest*, model *neural network* juga memiliki tingkat akurasi yang sangat tinggi, tetapi memerlukan *computing power*, memori, serta waktu pembelajaran yang lama. Terakhir, *support vector machine* memiliki akurasi yang cukup tinggi pula, tetapi fungsi *kernel* dari algoritma klasifikasinya sangat bergantung pada jenis data yang dimodelkan.

Pada kasus *image classification* yang menjadi objek utama dari makalah ini, penulis akan menggunakan model *decision tree* dikarenakan kompleksitasnya yang relatif rendah, tingkat akurasi yang mencukupi, serta konektivitas algoritma dan struktur data yang dimilikinya terharap teori yang diajarkan pada mata kuliah IF2120 Matematika Diskrit.

II. Landasan Teori

A. Definisi dari Struktur Data Tree

Tree adalah sebuah struktur data nonlinear (data tidak disimpan pada sebuah "rantai" lurus) dan memiliki hierarki yang jelas di antara elemen-elemennya (berupa *level* atau tingkatan). Setiap elemen pada *tree* (disebut juga *node*) mengandung nilai (*values*) dan beberapa pointer ke elemen "anaknya". Struktur data tree juga memiliki hubungan dengan struktur data *graph*, dimana *tree* adalah *graph* tidak berarah terhubung yang tidak mengandung sirkuit. Contoh dari sebuah struktur data *tree* adalah sebagai berikut:

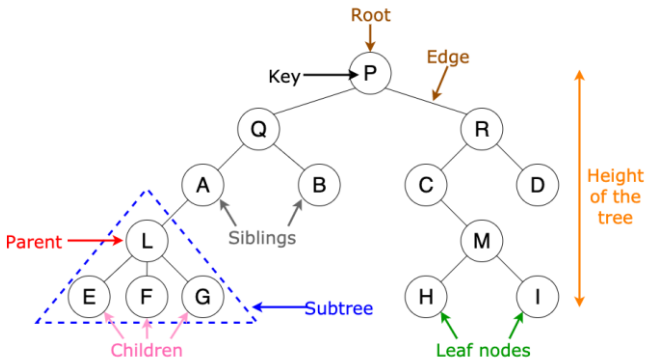


Gambar 2.1 Contoh struktur data *tree*
Sumber: Java T Point

Secara sekilas, struktur data *tree* memiliki struktur yang mirip dengan struktur data *graph*. Hal ini dikarenakan *tree* merupakan bentuk *graph* dengan beberapa syarat dan ketentuan. Akibatnya, struktur data *tree* memiliki beberapa properti yang tidak miliki oleh *graph*, diantaranya adalah:

1. Seperti *linked list*, struktur data *tree* memiliki properti rekursivitas dalam skema pemrosesannya. Bagian basis dari struktur data *tree* adalah *node* teratas dari *tree* tersebut, yang dinamakan *root node*. Bagian rekurens dari struktur data *tree* adalah *node* lainnya yang merupakan "keturunan" dari *root node* tersebut.
2. Setiap pasang simpul pada struktur data *tree* terhubung dengan satu sama lain dan hanya terdapat lintasan tunggal yang menghubungkan kedua *node* tersebut.
3. Struktur data *tree* tidak mengandung sirkuit (lintasan yang dimulai dan berakhir pada *node* yang sama tanpa melewati *edge* yang sama).
4. Jika sebuah struktur data *tree* memiliki n buah *node*, maka akan terdapat $n - 1$ buah *edge*.
5. Jika ditambahkan satu buah *edge* baru tanpa penambahan *node* baru, maka struktur data tersebut tidak akan menjadi *tree* dikarenakan adanya sirkuit.
6. Bilangan kromatis dari sembarang *tree* adalah 2 (warna *node* antara orangtua dengan anak harus berbeda).

B. Terminologi dari Struktur Data Tree



Gambar 2.2 Visualisasi terminologi *tree*
Sumber: towards data science

Beberapa istilah dan terminologi dari struktur data *tree* adalah diantaranya:

1. Anak (*child* atau *children*) dan Orangtua (*parent*)
   E, *F*, dan *G* adalah anak-anak dari node *L*, *L* adalah orangtua dari anak-anak itu.
2. Lintasan (*path*)
   Lintasan dari *P* ke *E* adalah *P*, *Q*, *A*, L, E. Panjang lintasan dari *P* ke *E* adalah 5.
3. Saudara kandung (*sibling*)
   A adalah saudara kandung *B*, tetapi *A* bukan saudara kandung *C*, dikarenakan orangtua mereka berbeda.
4. Upapohon (*subtree*)
   Upapohon (*L* (*E* () *F* () *G* ())) merupakan upapohon dari pohon pada contoh.
5. Derajat (*degree*)
   Derajat sebuah *node* adalah jumlah dari upapohon atau anak pada *node* tersebut. Sebagai contoh, derajat dari *L* adalah 3 dan derajat dari *R* adalah 2.
6. Daun (*leaf*)
   Daun adalah *node* yang berderajat nol (tidak mempunyai anak). Sebagai contoh, *E*, *F*, *G*, *B*, *H*, *I*, dan *D* adalah daun dari pohon.
7. Simpul dalam (*internal nodes*)
   Simpul dalam adalah *node* yang mempunyai anak (berderajat lebih dari nol). Sebagai contoh, *P*, *Q*, dan *A* adalah simpul dalam.
8. Tingkatan (*level*)
   Panjang lintasan dari *root node* ke *node* tersebut. Sebagai contoh, tingkatan dari node *A* adalah 2.
9. Tinggi (*height*)
   Tinggi dari suatu *tree* adalah lintasan terpanjang dari *root node* menuju *leaf node*. Tinggi dari *tree* pada contoh adalah 4.

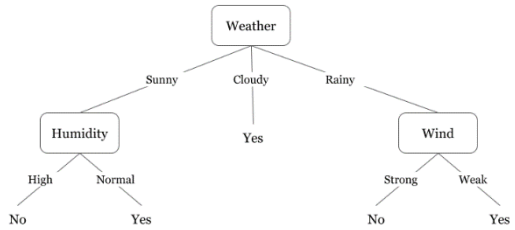
C. Definisi dari Decision Tree

Decision tree adalah salah satu model *supervised machine learning* yang dapat diaplikasikan ke berbagai jenis data. *Decision tree* bekerja dengan mencari cara untuk membagi data menjadi beberapa bagian dengan cara mengoptimisasikan parameter kondisi pembagian. Algoritma pembagian kasus dan parameter kondisi apa saja yang menyebabkan pembagian optimum akan di bahas pada bagian selanjutnya. *Decision tree* biasa digunakan untuk melakukan pengklasifikasian suatu data diskrit, meskipun juga dapat digunakan untuk regresi data kontinu. *Decision tree* juga memiliki kecenderungan untuk *overfitting*, dimana akurasi model tinggi ketika menggunakan *data training*, tetapi ketika dimasukkan data baru akan menjadi tidak akurat. Hal ini dikarenakan *decision tree* bekerja dengan mencoba membuat aturan baru hasil inferensi data *training*. Oleh karena itu, data *training* yang dimasukkan haruslah universal dan seimbang (frekuensi data tersebar dengan rata).
Berikut ini adalah contoh pembentukan *decision tree* dengan menggunakan analisis inferensi data masukan. Misalkan terdapat tabel yang berisi data hari, cuaca, temperatur, kelembapan, kekuatan angin, dan pertanyaan "Apakah hari ini bisa bermain tenis?".

Day	Weather	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Tabel 2.1 Tabel Data Inferensi untuk *Decision Tree*
Sumber: hackerearth

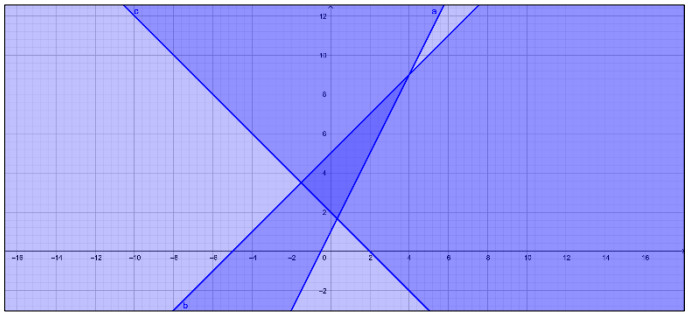
Decision tree dapat merepresentasikan data pada tabel tersebut dengan lebih efisien dikarenakan semua kemungkinan yang dapat dicapai akan direpresentasikan sebagai lintasan di dalam *tree*. Dengan menggunakan inferensi secara logika, dapat dibentuk *decision tree* sebagai berikut.



Gambar 2.3 *Decision tree* untuk pertanyaan "Apakah hari ini bisa bermain tenis?"
Sumber: hackerearth

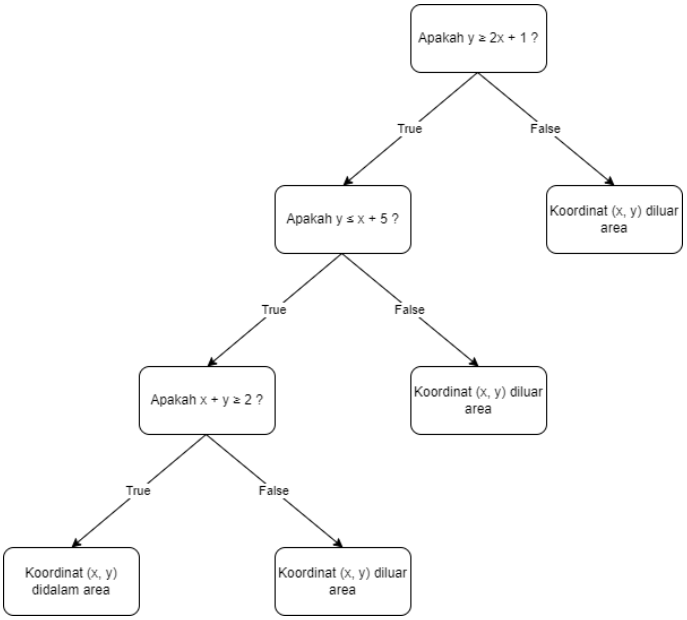
Selain menginferensikan data, *decision tree* juga dapat digunakan sebagai representasi fungsi boolean dalam bentuk apapun. Hal ini dilakukan dengan merepresentasikan *atomic* pada *node* dan nilai boolean dari *atomic* tersebut (*true* atau *false*) pada *edge decision tree*. Dengan begitu, *decision tree* dapat memodelkan data secara lebih komprehensif dibandingkan analisis regresi.
Decision tree juga dapat digunakan untuk membagi bidang berdimensi-*N* menjadi beberapa bagian. Biasanya, hal ini digunakan untuk membentuk *region* atau area dalam data dengan bentuk kontinu. Berikut ini adalah contoh dari bagaimana *decision tree* dapat membagi bidang kartesius.

Gambar di bawah merupakan sebuah area yang dibatasi oleh pertidaksamaan $y \geq 2x + 1$, $y \leq x + 5$, dan $x + y \geq 2$. Kemudian, akan dibuat *decision tree* untuk menentukan apakah koordinat (x, y) berasa di dalam area tersebut atau tidak.



Gambar 2.4 Koordinat kartesius dari area yang dibatasi oleh pertidaksamaan $y \geq 2x + 1$, $y \leq x + 5$, dan $x + y \geq 2$
Sumber: Dokumen penulis

Decision tree dapat dibuat dengan menginferensikan secara logika antara koordinat (x, y) sembarang dengan pertidaksamaan yang membatasi area di atas. Berikut ini merupakan gambar *decision tree* hasil inferensi data tersebut.



Gambar 2.5 *Decision tree* hasil inferensi koordinat kartesius
Sumber: Dokumen penulis

D. Algoritma Pembentukan Decision Tree

Algoritma pembentukan *decision tree* adalah algoritma yang digunakan untuk membentuk *decision tree* secara *top-down* dengan menggunakan data masukan sebagai inferensi. Algoritma ini menggunakan dua konsep yang sering digunakan dalam konsep algoritma, yaitu *greedy algorithm* dan *divide and conquer*.

Greedy algorithm adalah algoritma yang solusi optimalnya dapat diturunkan dari solusi optimal subpersoalannya serta pada setiap subpersoalan tersebut, akan terdapat langkah dimana langkah tersebut menghasilkan solusi optimal pada subpersoalan. Algoritma pembentukan *decision tree* disebut *greedy algorithm* dikarenakan algoritma selalu memilih partisi dengan nilai *information gain* (akan dijelaskan nanti) paling tinggi diantara semua kemungkinan pembagian. Hal tersebut akan menghasilkan *decision tree* dengan akurasi paling tinggi.

Divide and conquer adalah suatu cara atau metode penyeselaian masalah dengan cara membagi masalah tersebut menjadi beberapa submasalah yang lebih sederhana, kemudian menyelesaikan masalah tersebut, lalu menggabungkannya kembali menjadi solusi yang utuh. Umumnya, algoritma dengan metode *divide and conquer* adalah algoritma yang mengolah atau memproses struktur data rekursif, seperti *tree*, *heap*, *linked list*, *disjoint set*, dan lain-lain. Algoritma pembentukan *decision tree* disebut algoritma yang menggunakan metode *divide and conquer* dikarenakan adanya *recursive partitioning* pada pemrosesan setiap *node* dari *tree* tersebut. Kemudian, hasil-hasil partisi tersebut akan direkursif terus-menerus hingga tidak bisa dilakukan partisi kembali atau data yang dikelompokkan oleh *node* tersebut sudah seragam.

Untuk menentukan partisi yang paling optimum pada suatu *node*, secara *greedy* dapat diturunkan bahwa setiap data harus "homogen" dengan data lainnya pada partisi tersebut. Tolak ukur kehomogenan suatu partisi dapat dihitung dengan menggunakan fungsi *estimate of positive correctness and true positive rate*, *gini impurity*, *entropy*, *variance reduction*, atau *measure of goodness*. Pada makalah ini, hanya akan dijelaskan fungsi *gini impurity* dan *entropy* dikarenakan perhitungannya yang tidak terlalu kompleks dan terdapat tolak ukur yang jelas (tidak didefinisikan oleh pemrogram).

1. *Gini Impurity*

   Gini impurity adalah suatu parameter yang dapat menentukan tingkat kehomogenan suatu data dengan menghitung berapa peluang suatu elemen acak pada data diberi *class* atau label yang salah. Semakin tinggi nilai *gini impurity*, maka peluang sebuah elemen diberi *class* atau label yang tidak tepat akan semakin tinggi. Misalkan terdapat suatu data dengan jumlah *class* atau label sebanyak J dan p_i adalah data dengan *class* atau label i dimana $i \in \{1,2,3,\dots,J\}$, maka nilai *gini impurity* dari setiap partisi (T) tersebut dapat dihitung dari rumus:

$$H(T) = I_G(p) = 1 - \sum_{i=1}^{J} {p_i}^2$$

Persamaan 2.1 *Gini Impurity* dari satu partisi

2. *Entropy*

   Entropy adalah suatu parameter yang dapat menentukan tingkat kehomogenan suatu data dengan mengukur *degree of randomness* dari suatu data. Semakin tinggi nilai *entropy*, maka *degree of randomness* dari *dataset* tersebut semakin tinggi. Misalkan terdapat suatu data dengan jumlah *class* atau label sebanyak J dan p_i adalah data dengan kelas atau label i dimana $i \in \{1,2,3,\dots,J\}$, maka nilai *entropy* dari setiap partisi (T) tersebut dapat dihitung dari rumus:

$$H(T) = I_E(p) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

Persamaan 2.2 *Entropy* dari satu partisi

Dengan mengetahui nilai kehomogenan setiap partisi, maka bisa didefinisikan suatu parameter *information gain* yang berfungsi sebagai nilai kuantisasi informasi yang diperoleh dari partisi tersebut. Semakin tinggi nilai *information gain*, maka semakin banyak pula informasi yang kita peroleh akibat partisi tersebut. Misalkan terdapat suatu data yang dipartisi menggunakan fungsi f menjadi N bagian dan T_i adalah hasil partisi-partisinya tersebut dengan $i \in \{1,2,3,\dots,N\}$, maka nilai *information gain* dari hasil partisi tersebut adalah:

$$IG(T,f) = I_E(T) - \frac{1}{N}\sum_{i=1}^{N} I_E(T_i)$$

Persamaan 2.3 *Information Gain* dari hasil partisi

E. Algoritma Pemrosesan Data Image

Diketahui bahwa data di dalam sebuah *image* dapat direpresentasikan dalam bentuk matriks dua dimensi yang diisi oleh *pixel* dengan *color space* tertentu, seperti RGB, RGBA, dan lain-lain. Oleh karena itu, jumlah parameter yang harus disediakan oleh *decision tree* menjadi sangat banyak, yang akan berefek pada tingkat akurasi dan efektivitas dari algoritma *decision tree*. Selain itu, dikarenakan *size* serta *color space* dari setiap *image* berbeda-beda, maka jumlah parameter yang harus diproses oleh *decision tree* akan menjadi berbeda-beda pula. Akibatnya, perlu adanya standarisasi jumlah parameter. Salah satu cara mengatasi permasalahan ini adalah dengan melakukan metode *feature extraction*. *Features* itu sendiri dalam konteks *image processing* adalah sebuah nilai kuantitatif yang dapat berupa *integer*, *float*, *binary* atau sekumpulan nilai kuantitatif yang dapat berupa *list* atau vektor.

Ketika menentukan *features* mana yang akan digunakan, perlu diperhatikan bahwa agar *decision tree* bekerja secara akurat, *features* yang diambil haruslah mencakup semua *features* global *image* (*features* yang mengkuantisasi keseluruhan *pixel image*), seperti warna, tekstur, bentuk, dan lain-lain. Beberapa contoh algoritma *global features extraction* adalah *Color Channel Statistics*, *Color Histogram*, *Hu Moments*, *Zernike Moments*, *Haralick Textures*, *Local Binary Pattern*, *Histogram of Oriented Gradients*, dan *Threshold Adjacency Statistics*. Selain *features global*, dapat juga menambahkan *features* lokal (*features* yang mengkuantisasi sebagian *pixel image*) untuk menambah parameter pengidentifikasi *patches* atau *region* dari *image*. Beberapa contoh algoritma dari *local features extraction* adalah *Scale Variant Feature Transform*, *Speeded Up Robust Features*, *Oriented Fast and Rotated BRIEF*, dan *Binary Robust Independent Elementary Features*.

Ketika program menggunakan beberapa *features extraction*, maka pemrogram harus mencari cara untuk menggabungkan semua *features* itu ke dalam satu nilai atau struktur data sebagai parameter pada *decision tree*. Umumnya ketika menggabungkan beberapa *global features*, pemrogram dapat mengkonkatenasi *list*, vektor, dan/atau nilai numerik ke dalam suatu vektor panjang. Vektor panjang tersebut umumnya dinamakan *global feature vector*. Urutan konkatenasi pada vektor tersebut bebas (urutannya tidak mempengaruhi tingkat efisiensi dan akurasi *decision tree*). Tetapi, untuk penggabungan antara *local features* dengan *local features* lainnya atau *local features* dengan *global features*, maka diperlukan suatu metode yang dinamakan *Bag of Visual Words*. Pada makalah ini, hanya akan digunakan *global features* agar memudahkan pada bagian implementasi. *Global features* yang dipakai pada implementasi program makalah ini adalah *Color Histogram* untuk mengkuantisasi warna dari *image*; *Hu Moments* untuk mengkuantisasi bentuk objek dari *image*; serta *Haralick Textures* untuk mengkuantisasi tekstur dari *image*. Kemudian, semua hasil *features extraction* tersebut akan dikonkatenasi menjadi sebuah vektor.

III. Implementasi Decision Tree Image Classifier dalam Bahasa Python

A. Struktur Program

Struktur *directory tree* dari *source code* implementasi program adalah sebagai berikut.

```
.
├── global_var.py
├── main.py
├── output_tree.py
├── train_test.py
├── dataset
│   ├── test
│   │   ├── image_1.jpg
│   │   ├── image_2.jpg
│   │   └── ...
│   └── train
│       ├── label_1
│       │   ├── image_1.jpg
│       │   ├── image_2.jpg
│       │   └── ...
│       ├── label_2
│       │   ├── image_1.jpg
│       │   ├── image_2.jpg
│       │   └── ...
│       ├── ...
│       └── label_N
│           ├── image_1.jpg
│           ├── image_2.jpg
│           └── ...
└── output
    ├── data.h5
    └── labels.h5
```

Gambar 3.1 *Directory tree* program
Sumber: Dokumen penulis

Dapat dilihat dari gambar diatas, *directory program* terdiri atas empat *file source code* Python serta folder dataset. Keempat *file source code* Python tersebut adalah global_var.py, train_test.py, output_tree.py, serta main.py yang mengandung variabel, fungsi, prosedur, serta alur jalan program. Kemudian, terdapat folder dataset yang berisi subfolder lainnya, yaitu test dan train. Folder train mengandung subfolder yang berisi data masukan *image* dengan label atau *class* berdasarkan nama subfolder tersebut. Folder test berisi *image* yang akan dilakukan *prediction* oleh model *decision tree* yang telah dibuat menggunakan image dari folder train. Untuk mengakses *source code* selengkapnya, dapat diakses menggunakan pranala berikut ini.

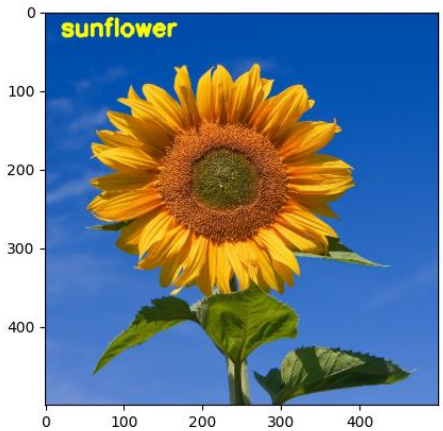
B. Implementasi Subprogram Image Preprocessing

File global_var.py mengandung deklarasi variabel awal, seperti *directory* dari folder dataset, jumlah *image* per *class*, serta *size* dari image setelah dikompresi. Selain itu, *file* global_var.py juga mengandung deklarasi dan definisi fungsi

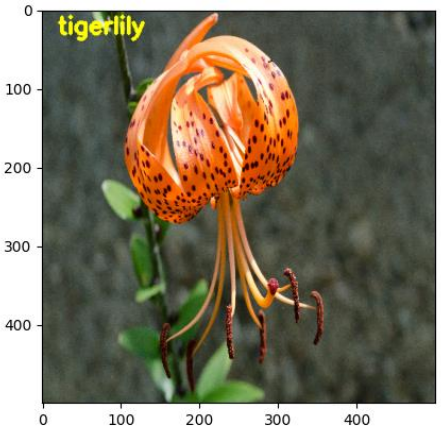
dan prosedur yang digunakan untuk melakukan *data preprocessing* pada *image data training*, seperti melakukan *resizing* pada *image*, melakukan *feature extraction*, menambahkan label sesuai nama folder dari *image* tersebut, dan kemudian menyimpan data hasil *preprocessing* tersebut pada folder output. Modul-modul eksternal yang digunakan pada subprogram ini adalah cv2 (mengandung fungsi dan prosedur yang digunakan untuk melakukan *image processing*), numpy (mengandung deklarasi dan definisi dari struktur data numpy *array*, yang digunakan untuk mengakses *array* yang dibentuk oleh modul cv2), mahotas (mengandung deklarasi dan definisi dari fungsi *haralick textures*), serta h5py (mengandung fungsi dan prosedur untuk melakukan *I/O stream* terhadap *file* dengan *extension* .h5 pada folder output).

C. Implementasi Subprogram Training dan Testing

File train_test.py mengandung deklarasi variabel yang diperlukan ketika eksekusi algoritma *training* dan *testing*, seperti besar persentase *test size* dari seluruh data masukan, *directory* dari folder dataset, *seed* yang digunakan dalam algoritma pembentukan *decision tree*, serta parameter *decision tree* yang dioptimumkan. Selain itu, *file* train_test.py juga mengandung deklarasi dan definisi fungsi dan prosedur yang digunakan untuk melakukan *training* dengan menggunakan data yang telah dilakukan *preprocessing* pada *file* global_var.py yang bertujuan untuk membuat *decision tree* serta *testing* dengan menggunakan *image* yang terdapat pada folder test yang bertujuan untuk melakukan *prediction* oleh model *decision tree* yang telah dibuat sebelumnya. Modul-modul eksternal yang digunakan pada subprogram ini adalah cv2 (mengandung fungsi dan prosedur yang digunakan untuk melakukan *image processing*), numpy (mengandung deklarasi dan definisi dari struktur data numpy *array*, yang digunakan untuk mengakses *array* yang dibentuk oleh modul cv2), h5py (mengandung fungsi dan prosedur untuk melakukan *I/O stream* terhadap *file* dengan *extension* .h5 pada folder output), sklearn (mengandung fungsi dan prosedur untuk membagi *dataset* menjadi *training dataset* dan *testing dataset* yang berfungsi untuk memberikan perkiraan persentase tingkat keakurasian), serta matplotlib (mengandung fungsi dan prosedur untuk menampilkan *verdict* hasil *prediction model* kepada pengguna menggunakan GUI).



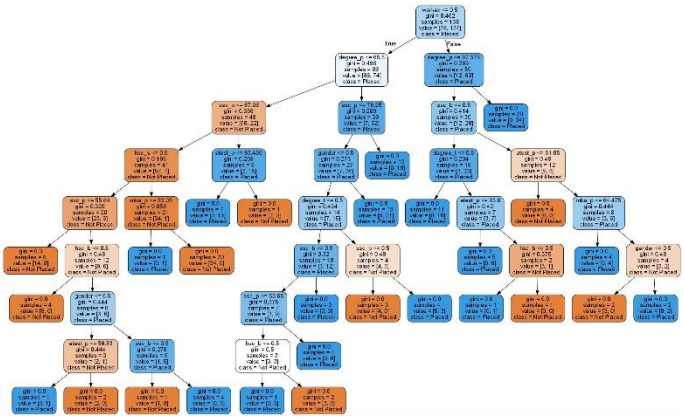
Gambar 3.2 Contoh hasil *prediction* menggunakan algoritma *decision tree*
Sumber: Dokumen penulis



Gambar 3.3 Contoh hasil *prediction* menggunakan algoritma *decision tree*
Sumber: Dokumen penulis

D. Implementasi Subprogram Display Tree

File output_tree.py mengandung deklarasi prosedur untuk melakukan penyimpanan skema struktur data *decision tree* dalam bentuk *image*. Kegunaan prosedur ini hanyalah untuk melakukan visualisasi *decision tree*, bukan untuk menyimpan model yang telah dibuat sebelumnya. Modul-modul eksternal yang digunakan pada subprogram ini adalah sklearn (mengandung fungsi dan prosedur untuk mengubah struktur data *decision tree* menjadi data dalam struktur .dot) serta pydot (mengandung fungsi dan prosedur untuk mengubah struktur data dalam bentuk .dot menjadi *image*). Selain modul, diperlukan pula *software* Graphviz yang digunakan secara terintegrasi oleh modul pydot untuk memvisualisasikan *graph* yang dapat diunduh menggunakan pranala berikut ini. Berikut merupakan salah satu contoh *image* yang dihasilkan oleh subprogram.



Gambar 3.3 Contoh hasil visualisasi *decision tree* dalam bentuk *image*
Sumber: Dokumen penulis

E. Implementasi Program Utama

File main.py mengandung instruksi alur utama program serta merupakan *driver* dari seluruh *file* Python lainnya. Terdapat empat *query* yang dapat dilakukan pengguna untuk membuat model *decision tree*, yaitu PREPARE, TRAIN, TEST, serta QUIT. *Query* PREPARE akan mengeksekusikan subprogram prepare_data, yang bertujuan untuk melakukan *data preprocessing* pada *image training*. Kemudian, *query* TRAIN akan mengeksekusikan subprogram train_model, yang bertujuan untuk membuat model *decision tree* dengan *image training* pada folder train sebagai inferensi. Selanjutnya, *query* TEST akan mengeksekusikan subprogram test_model, yang bertujuan untuk melakukan *prediction* terhadap *image testing* pada folder test menggunakan model yang telah dibuat sebelumnya dan kemudian menampilkan hasil *verdict* dari *image* tersebut. Terakhir, *query* QUIT akan menghentikan eksekusi, melakukan terminasi, dan keluar dari program. Modul-modul eksternal yang digunakan pada subprogram ini adalah sklearn (mengandung deklarasi dan definisi dari struktur data *decision tree* yang merupakan objek utama dalam algoritma *training* dan *testing*).

IV. KESIMPULAN

Decision tree merupakan salah satu model *supervised machine learning* yang umumnya digunakan untuk melakukan klasifikasi terhadap data masukan. *Decision tree* memiliki bentuk struktur data rekursif dalam bentuk *tree* dengan *node* merepresentasikan suatu pertanyaan atas parameter data dan *edge* merepresentasikan semua kemungkinan jawaban atas pertanyaan tersebut. *Decision tree* dapat melakukan *prediction* atas banyak jenis data, seperti data boolean, numerik, *string*, maupun vektor. Algoritma pembentukan *decision tree* menggunakan konsep *top-down greedy algorithm* (digunakan untuk pencarian *decision tree* dengan akurasi paling besar menggunakan *information gain*) serta *divide and conquer* (mempartisi data menjadi bagian yang lebih mudah untuk diklasifikasi lebih lanjut). Pada pemrosesan *image*, digunakan metode *features extraction* untuk memperoleh *global features vector* yang berdimensi sama untuk setiap *size* pada *image*. Pada bagian implementasi *source code* program, digunakan beberapa modul Python seperti sklearn, numpy, dan cv2 untuk membantu pemrogram dalam melakukan *image preprocessing*, *model training*, *model testing*, dan *model visualization*.

V. UCAPAN TERIMA KASIH

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan kasih karunia-Nya yang telah memberikan kesehatan dan kesempatan kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih sebesar-besarnya kepada seluruh pihak yang telah membantu penulis dalam menyelesaikan makalah ini, diantaranya adalah:

1. Dra. Harlili, M.Sc. selaku dosen pengampu kelas K02 mata kuliah IF2120 Matematika Diskrit.
2. Dr. Ir. Rinaldi, M.T. atas kontribusi buku dan materi yang penulis kutip pada makalah ini.
3. Seluruh pihak yang telah membuat *source code* dan modul secara *open-source* yang penulis kutip pada makalah ini.

REFERENSI

[1] St. George, Benjamin dan Alexander S. Gilis. 2021. *Turing Test*, https://searchenterpriseai.techtarget.com/definition/Turing-test, diakses pada 11 Desember 2021.

[2] United States Department of Transportation. *Automated Vehicles for Safety*, https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety, diakses pada 12 Desember 2021.

[3] Shwartz, Steve. 2021. *Are Self-Driving Cars Really Safer Than Human Drivers?*, https://thegradient.pub/are-self-driving-cars-really-safer-than-human-drivers/, diakses pada 12 Desember 2021.

[4] Munir, Rinaldi. 2021. Pohon (Bag. 1), https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf, diakses pada 12 Desember 2021.

[5] Munir, Rinaldi. 2021. Pohon (Bag. 2), https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf, diakses pada 12 Desember 2021.

[6] Jain, Sidhi. 2021. *Introduction to Tree Data Structure*, https://www.geeksforgeeks.org/introduction-to-tree-data-structure/, diakses pada 12 Desember 2021.

[7] JavaTpoint. *Tree Data Structures*, https://www.javatpoint.com/tree, diakses pada 12 Desember 2021.

[8] Aji, Alham Fikri dan Wiliam Gozali. Pemrograman Kompetitif Dasar, https://ksn.toki.id/data/pemrograman-kompetitif-dasar.pdf, diakses pada 12 Desember 2021.

[9] Mallawaarachchi, Vijini. 2020. *8 Useful Tree Data Structures*, https://towardsdatascience.com/8-useful-tree-data-structures-worth-knowing-8532c7231e8c, diakses pada 12 Desember 2021.

[10] Raj, Ashwin. 2020. *Sowing the Seeds of Decision Tree Regression*, https://towardsdatascience.com/sowing-the-seeds-of-decision-tree-regression-2bb238dfd768, diakses pada 13 Desember 2021.

[11] Galarnyk, Michael. 2020. *Visualizing Decision Tree with Python (Scikit-learn, Graphviz, Matpotlib)*, https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc, diakses pada 10 Desember 2021.

[12] Raj, Ashwin. 2021. *An Exhaustive Guide to Decision Tree Classification in Python 3.x*, https://towardsdatascience.com/an-exhaustive-guide-to-classification-using-decision-trees-8d472e77223f, diakses pada 10 Desember 2021.

[13] Shubham. *Decision Tree*, https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/ml-decision-tree/tutorial/, diakses pada 12 Desember 2021.

[14] Ilango, Gogul. 2017. *Image Classification using Python and Scikit-learn*, https://gogul.dev/software/image-classification-python, diakses pada 10 Desember 2021.

[15] Nislback, Maria-Elena dan Andrew Zissserman. *17 Category Flower Dataset*, https://www.robots.ox.ac.uk/~vgg/data/flowers/17/, diakses pada 10 Desember 2021.

[16] Scikit Learn Development and Maintenance. 2021. *Scikit-learn Documentation*, https://scikit-learn.org/stable/index.html, diakses pada 10 Desember 2021.

[17] The NumPy Community. 2021. *NumPy v1.21 Manual*. https://numpy.org/doc/stable/index.html, diakses pada 10 Desember 2021.

[18] Banerjee, Prashant. 2019. *Decision-Tree Classifier Tutorial*, https://www.kaggle.com/prashant111/decision-tree-classifier-tutorial, diakses pada 10 Desember 2021.

[19] Navlani, Avinash. 2018. *Decision Tree Classification in Python*, https://www.datacamp.com/community/tutorials/decision-tree-classification-python, diakses pada 10 Desember 2021.

[20] Dicoding Indonesia. Belajar *Machine Learning* untuk Pemula, https://www.dicoding.com/academies/184, diakses pada 10 Desember 2021.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Desember 2021

Rayhan Kinan Muhannad 13520065