BAB IV KLASIFIKASI

Klasifikasi merupakan penempatan objek-objek ke salah satu dari beberapa kategori yang telah ditetapkan sebelumnya. Klasifikasi telah banyak ditemui dalam berbagai aplikasi. Sebagai contoh, pendeteksian pesan email spam berdasarkan *header* dan isi atau mengklasifikasikan galaksi berdasarkan bentuk-bentuknya. Dalam bab ini akan dibahas mengenai konsep klasifikasi, beberapa isi penting dalam klasifikasi dan menyatakan metode untuk mengevaluasi dan membandingkan kinerja teknik klasifikasi.

4.1 Model Klasifikasi

Data input untuk klasifikasi adalah koleksi dari *record*. Setiap *record* dikenal sebagai *instance* atau contoh, yang ditentukan oleh sebuah *tuple* (**x**, y), dimana x adalah himpunan atribut dan y adalah atribut tertentu, yang dinyatakan sebagai label kelas (juga dikenal sebagai kategori atau atribut target). Tabel 4.1 menunjukkan contoh *data set* yang digunakan untuk mengklasifikasikan vertebrata ke dalam salah satu dari kategori berikut: mammal, bird, fish, reptile atau amphinian. Himpunan atribut meliputi sifat-sifat dari vertebrata seperti suhu tubuh, permukaan kulit, metode reproduksi, kemampuannya untuk terbang, dan kemampuannya untuk hidup di air. Walaupun atribut yang dinyatakan dalam Tabel 4.1 kebanyakan diskret, himpunan atribut tersebut juga dapat mengandung fitur kontinu. Label kelas harus merupakan atribut diskret. Klasifikasi berbeda dengan regresi, dimana regresi merupakan pemodelan prediktif dimana y adalah atrubut kontinu.

Tabel 4.1 *Data set* vertebrata

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

Definisi 4.1 (Klasifikasi):

Klasifikasi adalah tugas pembelajaran sebuah fungsi target f yang memetakan setiap himpunan atribut **x** ke salah satu label kelas y yang telah didefinisikan sebelumnya.

Fungsi target juga dikenal secara informal sebagai model klasifikasi. Model klasifikasi berguna untuk keperluan berikut:

Pemodelan Deskriptif. Model klasifikasi dapat bertindak sebagai alat penjelas untuk membedakan objek-objek dari kelas-kelas yang berbeda. Sebagai contoh, untuk para akhir Biologi, model deskriptif yang meringkas data seperti ditunjukkan dalam Tabel 4.1 dapat berguna untuk menjelaskan fitur-fitur apakah yang mendefinisikan vertebrata sebagai mammal, bird, fish, reptile atau amphinian.

Pemodelan Prediktif. Model klasifikasi juga dapat digunakan untuk memprediksi label kelas dari *record* yang tidak diketahui. Seperti yang ditunjukkan dalam Tabel 4.2, sebuah model klasisfikasi dapat dipandang sebagai kotak hitam yang secara otomatis memberikan sebuah label kelas ketika dipresentasikan dengan himpunan atribut dari *record* yang tidak diketahui.

Gambar 4.2 Klasifikasi sebagai pemetaan sebuah himpunan atribut input **x** ke dalam label kelasnya y.

Anggaplah diberikan karakteristik dari sebuah mahluk yang dikenal sebagai monster gila:

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
gila monster	cold-blooded	scales	no	no	no	yes	yes	?

Kita dapat menggunakan model klasifikasi yang dibangun dari *data set* yang ditunjukkan dalam Tabel 4.1 untuk menentukan kelas yang sesuai untuk mahluk tersebut.

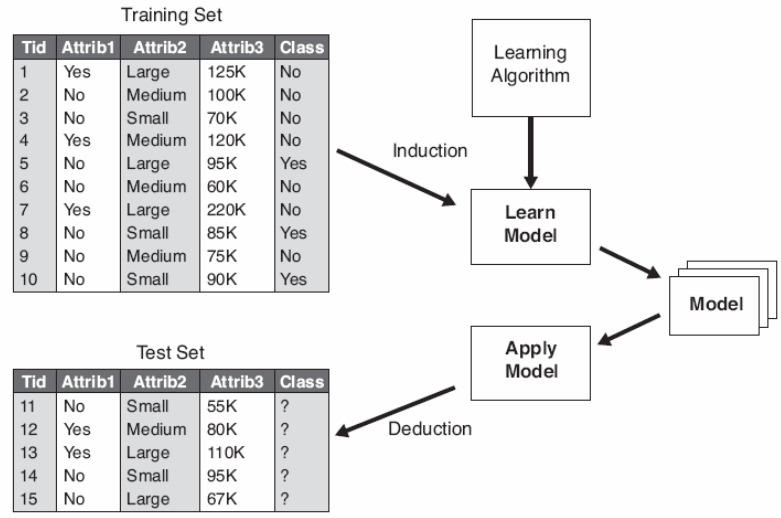
Teknik klasifikasi seringkali digunakan untuk memprediksi atau mendeskripsikan *data set* dengan kategori biner dan nominal. Teknik klasifikasi kurang efektif untuk kategori ordinal (seperti untuk mengklasifikasikan seseorang sebagai anggota dari kelompok dengan income tinggi, sedang atau rendah) karena teknik-teknik ini tidak memperhatikan urutan implisit diantara kategori. Bentuk-bentuk hubungan lain, seperti hubungan *subclass-superclass* diantara kategori, (contoh, manusia dan siamang adalah primata, yang merupakan *subclass* dari mammal) dapat diabaikan.

4.2 Pendekatan Umum untuk Menyelesaikan Masalah Klasifikasi

Teknik klasifikasi (klasifier) adalah pendekatan sistematis untuk pembuatan model klasifikasi dari sebuah *data set* input. Contoh-contoh yang diberikan meliputi *decision tree classifier*, *rule-based classifier*, neural network, s*upport vector machines*, dan naive Bayes *classifier*. Setiap teknik menggunakan algoritme pembelajaran untuk mengidentifikasi model yang memberikan hubungan yang paling sesuai antara himpunan atribut dan label kelas dari data input. Model yang dibangun dengan sebuah algoritme pembelajaran haruslah sesuai dengan data input dan memprediksi dengan benar label kelas dari *record* yang belum pernah terlihat sebelumnya. Dengan demikian, kunci utama dari

algoritme pembelajaran adalah membangun model dengan kemampuan generalisasi yang baik, aitu model yang secara akurat memprediksi label kelas dari *record* yang tidak diketahui sebelumnya.

Gambar 4.3 menunjukan pendekatan umum untuk penyelesaian masalah klasifikasi. Pertama, ***training set*** berisi *record* yang mempunyai label kelas yang diketahui haruslah tersedia. *Training set* digunakan untuk membangun model klasifikasi, yang kemudian diaplikasikan ke ***test set***, yang berisi *record-record* dengan label kelas yang tidak diketahui.



Gambar 4.3 Pendekatan umum untuk pembangunan model klasifikasi.

Evaluasi dari kinerja model klasifikasi didasarkan pada banyaknya (*count*) *test record* yang diprediksi secara benar dan secara tidak benar oleh model. Count ini ditabulasikan dalam sebuah tabel yang dikenal sebagai *confusion matrix*. Tabel 4.2 menggambarkan *confusion matrix* untuk masalah klasifikasi biner. Setiap entri f_{ij} dalam tabel ini menyatakan banyaknya *record* dari kelas i yang diprediksi menjadi kelas j. Sebagai contoh, f_{01} adalah banyaknya *record* dari kelas 0 yang secara tidak benar diprediksi sebagai kelas 1. Berdasarkan pada entri-entri dalam *confusion matrix*, banyaknya total prediksi yang benar yang dibuat oleh model adalah ($f_{11} + f_{00}$) dan banyaknya total prediksi yang tidak benar adalah ($f_{10} + f_{01}$).

Tabel 4.2 *Confusion matrix* untuk masalah klasifikasi kelas

		Predicted Class	
		$Class = 1$	$Class = 0$
Actual Class	$Class = 1$	f_{11}	f_{10}
	$Class = 0$	f_{01}	f_{00}

Informasi dalam *confusion matrix* diperlukan untuk menentukan kinerja model klasifikasi. Ringkasan informasi ini ke dalam sebuah nilai digunakan untuk membandingkan kinerja dari model-model yang berbeda. Hal ini dapat dilakukan dengan menggunakan *performace metric* seperti **akurasi** yang didefiniskan sebagai berikut:

$$\text{Akurasi} = \frac{\text{banyaknya prediksi yang benar}}{\text{total banyaknya prediksi}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \quad (4.1)$$

Secara ekuivalen, kinerja sebuah model dapat dinyatakan dalam bentuk ***error rate*** –nya, yang diberikan oleh persamaan berikut:

$$\text{Error rate} = \frac{\text{banyaknya prediksi yang salah}}{\text{total banyaknya prediksi}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}} \quad (4.2)$$

Kebanyakan algoritme klasifikasi mencari model yang mencapai akurasi paling tinggi, atau secara ekuivalen *error* yang paling rendah ketika diaplikasikan ke *test set*.

4.3 *Decision tree* Induction

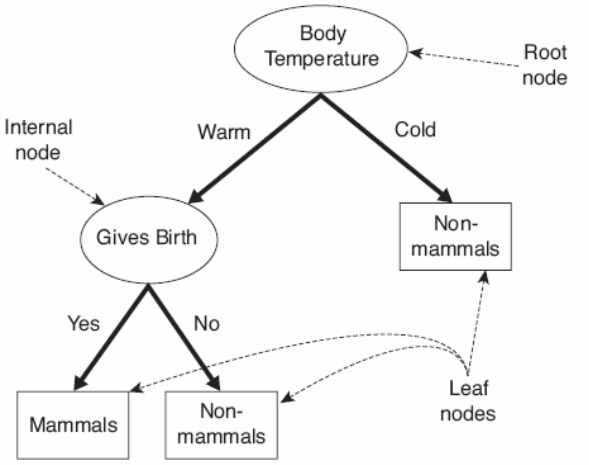
Klasifier pohon keputusan (*decision tree*) merupakan teknik klasifikasi yang sederhana yang banyak digunakan. Bagian ini membahas bagaimana pohon keputusan bekerja dan bagaimana pohon keputusan dibangun.

4.3.1 Cara Kerja Pohon Keputusan

Untuk memberikan ilustrasi dari cara kerja pohon keputusan, perhatikan masalah klasifikasi vertebrata dalam bentuk yang lebih sederhana. Vertebrata tidak diklasifikasikan ke dalam lima kategori yang berbeda, tetapi ke dalam dua kategori yaitu mammal dan non-mammal.

Anggaplah spesies baru ditemukan oleh peneliti dan ingin diketahui apakah spesies tersebut mammal atau non-mammal. Salah satu pendekatan yang dapat digunakan adalah dengan mengajukan serangkaian pertanyaan tentang karakteristik spesies. Pertanyaan pertama yang dapat diajukan adalah apakah spesies tersebut *cold* atau *warm-blooded*. Jika spesies tersebut *cold-blooded*, maka spesies tersebut bukan mammal. Selainnya dapat merupakan mammal atau bird. Pertanyaan selanjutnya yang dapat diajukan adalah apakah spesies betinanya melahirkan anak?, jika iya maka spesies tersebut adalah mammal, selainnya bukan mammal (kecuali untuk spesies mammal tertentu, *egg-laying mammal*).

Contoh tersebut mengilustrasikan bagaimana kita dapat menyelesaikan masalah klasifikasi dengan mengajukan serangkaian pertanyaan tentang atribut-atribut dari *test record*. Setiap kali jawaban diperoleh, pernyataan berikutnya diajukan sampai diperoleh sebuah kesimpulan mengenai label kelas dari *record*. Rangkaian pertanyaan tersebut dan jawaban-jawabannya yang mungkin dapat diorganisasikan ke dalam bentuk pohon keputusan, yang merupakan struktur hirarki yang terdiri dari *node-node* dan *edge-edge* berarah. Gambar 4.4 menunjukkan pohon keputusan untuk masalah klasifikasi mammal.



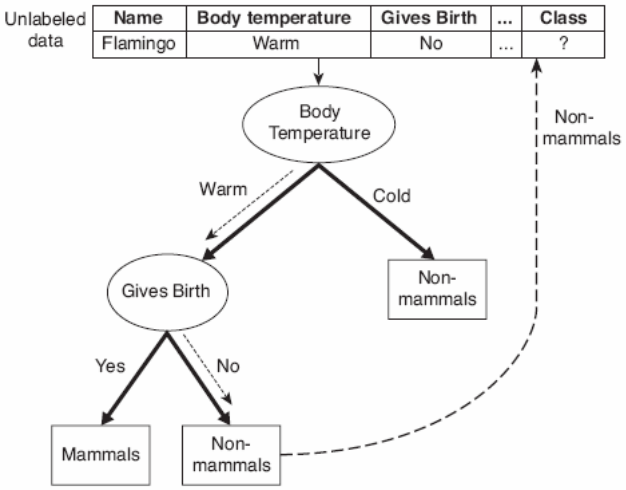
Gambar 4.4. Pohon keputusan untuk masalah klasifikasi mammal

Tree dalam Gambar 4.4 memiliki tiga bentuk *node*, yaitu:

- ***Root node***, yang tidak memiliki *edge* yang masuk dan memiliki nol atau lebih *edge* yang keluar.
- ***Internal node***, setiap *internal node* memiliki tepat satu *edge* yang masuk dan dua atau lebih *edge* yang keluar.
- ***Leaf*** atau ***terminal node***, setiap *terminal node* memiliki tepat satu *edge* yang masuk dan tidak ada *edge* yang keluar.

Dalam pohon keputusan, *leaf node* diberikan sebuah label kelas. Non-*terminal node*, yang terdiri dari *root* dan *internal node* lainnya, mengandung kondisi-kondisi uji atribut untuk memisahkan *record* yang memiliki karakteristik yang berbeda. Sebagai contoh, *root* pada Gambar 4.4 menggunakan atribut `Body Temperature` untuk memisahkan vertebrata berdarah panas (*warm-blooded*) dari vertebrata berdarah dingin (*cold-blooded*). Karena semua vertebrata berdarah dingin (*cold-blooded*) bukanlah mammal, sebuah *leaf node* yang diberi label non-mammal dibuat sebagai anak pada bagian kanan dari *root*. Jika vertebrata adalah berdarah panas (*warm-blooded*), maka atribut selanjutnya, `Gives Birth`, digunakan untuk membedakan mammal dari mahluk berdarah panas lainnya, yang kebanyakan adalah bird.

Setelah pohon keputusan dikonstruksi, *test record* dapat diklasifikasi. Bermula dari *root*, kondisi tes diaplikasikan ke *record* dan mengikuti cabang yang sesuai berdasarkan keluaran dari tes. Hal ini akan membawa kita ke *internal node* yang lain, dimana kondisi tes yang baru diaplikasikan, atau ke *leaf node*. Sebagai ilustrasi, Gambar 4.5 menunjukkan pergerakan *path* dalam pohon keputusan yang digunakan untuk memprediksi label kelas dari flamingo. *Path* berakhir pada *leaf node* dengan label `Non-mammals`. Garis putus-putus dalam Gambar 4.5 merepresentasikan keluaran dari penggunaan berbagai kondisi tes atribut dari vertebrata yang berlabel.



Gambar 4.5 Mengklasifikasi vertebrata yang tidak berlabel.

4.3.2 Membangun Pohon Keputusan

Secara prinsip terdapat banyak pohon keputusan yang secara eksponensial dapat dikonstruksi dari sekumpulan atribut yang diberikan. Beberapa *tree* lebih akurat dari *tree* yang lain, akan tetapi pencarian *tree* yang optimal secara komputasional tidak layak karena ukuran dari ruang pencarian adalah eksponensial. Algoritme yang efisien telah dibangun untuk mendapatkan pohon keputusan yang akurat dalam jumlah waktu yang layak. Algoritme ini biasanya menggunakan strategi *greedy* yang membuat pohon keputusan dengan membuat serangkaian keputusan optimum secara lokal mengenai atribut yang akan digunakan untuk mempartisi data. Salah satu algoritme demikian adalah Algoritme Hunt, yang merupakan dasar dari banyak algoritme induksi pohon keputusan yang telah ada, seperti ID3, C4.5 dan CART.

Algoritme Hunt

Dalam Algoritme Hunt, sebuah pohon keputusan berkembang secara rekursif dengan mempartisi *training record* ke dalam *subset-subset*. Misalkan D_t adalah himpunan dari *training record* yang berasosiasi dengan *node* t dan $y = \{y_1, y_2, ..., y_c\}$ adalah label-label kelas. Berikut adalah definisi rekursif dari Algoritme Hunt:

Langkah 1: Jika semua *record* dalam D_t anggota kelas yang sama y_t, maka t adalah *leaf node* dengan label y_t.

Langkah 2: Jika D_t mengandung *record* yang merupakan anggota dari lebih dari dari satu kelas, sebuah **kondisi tes atribut** dipilih untuk mempartisi *record-record* ke dalam *subset-subset* yang lebih kecil. *Child node* dibuat untuk setiap keluaran dari kondisi tes dan *record-record* dalam D_t didistribusikan ke *child*ren berdasarkan pada keluaran dari kondisi tes. Selanjutnya, algoritme secara rekursif diaplikasikan ke setiap *child node*.

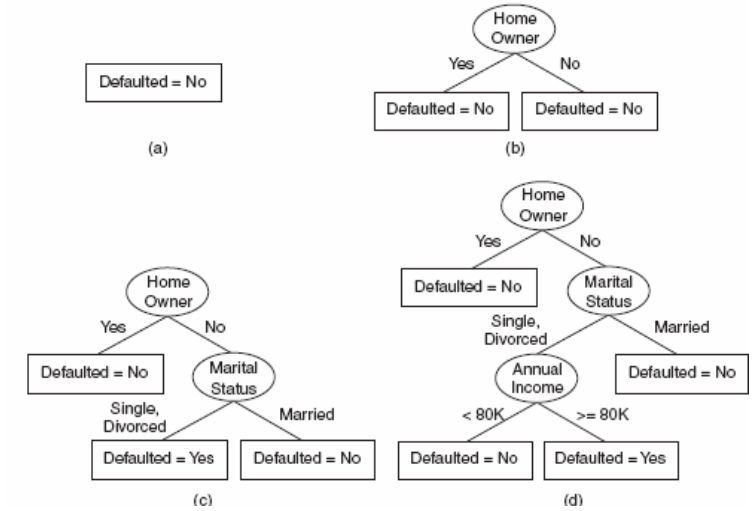
Untuk mengilustrasikan bagaimana algoritme ini bekerja, perhatikan masalah memprediksi apakah seorang pelamar pinjaman akan membayar kembali obligasi-obligasi pinjamannya ataukah menjadi penunggak, setelah itu melalaikan pinjamannya. Sebuah *training set* dari masalah ini dapat dikonstruksi dengan memeriksa *record-record* peminjam-peminjam sebelumnya. Dalam contoh yang

ditunjukkan dalam Gambar 4.6, setiap *record* mengandung informasi personal dari seorang pemijam bersama dengan label kelas yang menunjukkan apakah peminjam memiliki kelalaian dalam pembayaran pinjaman.

	binary	categorical	continuous	class
Tid	**Home Owner**	**Marital Status**	**Annual Income**	**Defaulted Borrower**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Gambar 4.6 *Training set* untuk memprediksi peminjam-peminjam yang akan melalaikan pembayaran pinjamannya.

Tree awal untuk masalah klasifikasi mengandung sebuah *node* dengan label kelas `Defaulted = No` (Gambar 4.7(a)), yang berarti kebanyakan dari para peminjam melakukan pembayaran pinjaman-pinjamannya. *Tree* perlu diperbaiki karena *root* mengandung *record-record* dari kedua kelas. Selanjutnya *record-record* dibagi ke dalam *subset-subset* yang lebih kecil berdasarkan pada keluaran dari kondisi tes `Home Owner`, seperti ditunjukkan dalam Gambar 4.7(b). Justifikasi pemilihan kondisi tes atribut akan dijelaskan kemudian. Selanjutnya Algoritme Hunt diaplikasikan secara rekursif ke setiap *child* dari *root*. Dari *training set* yang diberikan dalam Gambar 4.6, perhatikan bahwa semua peminjamn yang merupakan pemilik rumah membayar kembali pinjaman-pinjamannya. *Child* kiri dari *root* adalah *leaf node* yang diberi label `Defaulted = No` (Gambar 4.7(b)). Untuk *child* kanan, langkah rekursif dari algoritme Hunt perlu diaplikasikan kembali sampai semua *record* menjadi anggota kelas yang sama. *Tree* yang dihasilkan dari setiap langkah rekursif ditunjukkan dalam Gambar 4.7(c) dan (d).



Gambar 4.7 Algoritme Hunt untuk menghasilkan pohon keputusan

Algoritme Hunt akan bekerja jika setiap kombinasi dari nilai atribut dinyatakan dalam *training data* dan setiap kombinasi memiliki label kelas yang unik. Kondisi tambahan diperlukan untuk menangani kasus ini:

1. Beberapa *child node* yang dibuat dalam langkah 2 mungkin kosong; yaitu tidak ada *record* yang bersesuaian dengan *node-node* ini. Hal ini dapat terjadi jika tidak ada satupun *training record* yang memiliki kombinasi dari nilai atribut yang sesuai dengan *node-node* demikian. Dalam kasus ini *node* dinyatakan *leaf node* dengan label kelas yang sama seperti kelas dari *training record* yang berasosiasi dengan *node parent*-nya.
2. Dalam langkah 2, jika semua *record* yang berasosiasi dengan D_t memiliki nilai atribut yang identik (kecuali untuk label kelas), maka tidak dimungkinkan untuk memisahkan *record-record* ini lebih lanjut. Dalam kasus ini, *node* dinyatakan *leaf node* dengan label kelas yang sama seperti kelas dari *training record* yang berasosiasi dengan *node* ini.

Isu-isu Perancangan dari Induksi Pohon Keputusan

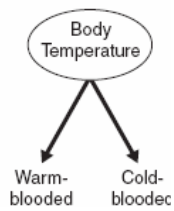
Sebuah algoritme pembelajaran untuk menghasilkan pohon keputusan harus memperhatikan dua isu berikut:

1. Bagaimana *training record* seharunya dipisah? Setiap langkah rekursif dari proses pertumbuhan *tree* harus memilih sebuah kondisi tes atribut untuk membagi *record-record* ke dalam *subset-subset* yang lebih kecil. Untuk mengimplementasikan langkah ini, algoritme harus menyediakan metode untuk menentukan kondisi tes untuk tipe atribut yang berbeda dan juga ukuran objektif untuk mengevaluasi setiap kondisi tes.
2. Bagaimana proses pemisahan berhenti? Kondisi berhenti diperlukan untuk menentukan proses pertumbuhan *tree*. Strategi yang mungkin adalah meneruskan perluasan sebuah *node* sampai semua *record* menjadi anggota kelas yang sama atau seluruh *record* memiliki nilai atribut yang identik. Kondisi ini merupakan kondisi cukup untuk menghentikan algoritme induksi pohon keputusan.

4.3.3 Metode untuk Menyatakan Kondisi Tes Atribut

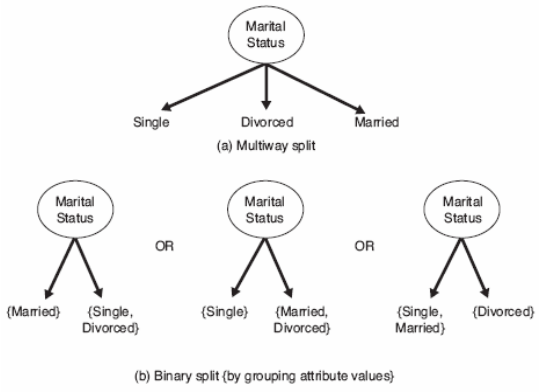
Algoritme induksi pohon keputusan haruslah menyediakan sebuah metode untuk menyatakan kondisi tes atribut dan keluaran yang sesuai untuk tipe-tipe atribut yang berbeda.

Atribut Biner. Kondisi tes untuk atribut biner menghasilkan dua keluaran yang potensial seperti ditunjukkan dalam Gambar 4.8.



Gambar 4.8 Kondisi tes untuk atribut biner

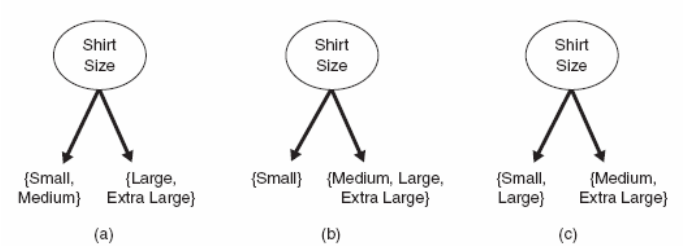
Atribut Nominal. Karena atribut nominal dapat memiliki banyak nilai, kondisi tes-nya dapat dinyatakan dalam dua cara, seperti ditunjukkan dalam Gambar 4.9. Untuk pemisahan *multiway* (*multiway* split) (Gambar 4.9(a)), banyaknya keluaran tergantung pada banyaknya nilai yang berbeda untuk atribut yang sesuai. Sebagai contoh, jika atribut seperti marital status memiliki tiga nilai yang berbeda, yaitu singel, married, atau divorced, kondisi tes-nya akan menghasilkan pemisahan tiga cara (three-way split). Beberapa algoritme pohon keputusan, seperti CART, hanya menghasilkan pemisahan biner dengan memperhatikan $2^{k-1} - 1$ cara pembuatan partisi biner dari k nilai atribut. Gambar 4.9(b) mengilustrasikan tiga cara pengelompokan nilai atribut untuk marital status ke dalam dua *subset*.



Gambar 4.9 Kondisi tes untuk atribut nominal

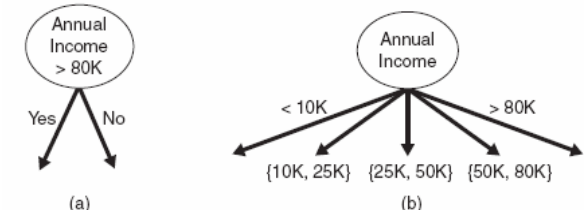
Atribut Ordinal. Atribut ordinal juga dapat menghasilkan pemisahan biner atau *multiway*. Nilai atribut ordinal dikelompokkan selama pengelompokan tidak melanggar sifat urutan dari nilai-nilai atribut. Gambar 4.10 mengilustrasikan berbagai cara pemisahan *training record* berdasarkan atribut `Shirt Size`. Pengelompokan yang ditunjukkan dalam Gambar 4.10(a) dan (b) mempertahankan urutan diantara nilai atribut, sebaliknya pengelompokan yang ditunjukkan dalam Gambar 4.10(c) merusak sifat urutan dari nilai-nilai atribut karena pengelompokkan mengkombinasikan nilai `Small` dan `Large` ke dalam

partisi yang sama sedangkan `Medium` dan `Extra Large` dikombinasikan ke dalam partisi yang lain.

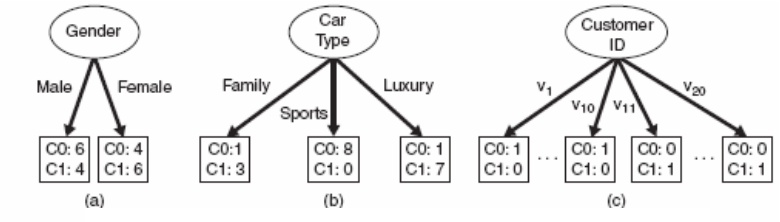


Gambar 4.10 Cara-cara pengelompokkan nilai atribut ordinal

Atribut Kontinu. Untuk atribut kontinu, kondisi tes dapat dinyatakan sebagai uji perbandingan (**A** < v) atau (**A** > v) dengan keluaran biner, atau kueri range dengan keluaran dalam bentuk $v_i \leq \mathbf{A} < v_{i+1}$, untuk i = 1, ..., k. Perbedaan antara kedua pendekatan ini diberikan dalam Gambar 4.11. Untuk kasus biner, algoritme pohon keputusan harus memperhatikan semua kemungkinan posisi pemisahan v, dan memilih salah satu yang menghasilkan partisi yang paling baik. Untuk pemisahan *multiway*, algoritme harus memperhatikan semua range nilai kontinu yang mungkin. Salah satu pendekatan yang dapat digunakan adalah strategi diskretisasi yang dijelaskan pada Bab 2. Setelah diskretisasi, nilai ordinal yang baru diberikan ke setiap interval yang didiskretkan. Interval-interval yang bersebelahan juga dapa diagregatkan ke dalam range yang lebih besar sepanjang sifat urutan dipertahankan.



Gambar 4.11. Kondisi tes untuk atribut kontinu



Gambar 4.12. Pemisahan *multiway* vs biner

4.3.4 Ukuran–ukuran untuk Pemilihan Pemisahan Terbaik

Terdapat banyak ukuran yang dapat digunakan untuk menentukan cara pemisahan *record* yang paling baik. Ukuran-ukuran ini didefinisikan dalam bentuk distribusi kelas dari *record* sebelum dan sesudah pemisahan.

Misalkan p(i|t) adalah fraksi dari *record* yang merupakan anggota dari kelas i pada sebuah *node* yang diberikan t. Kadang-kadang referensi ke *node* t dihilangkan dan menyatakan fraksi sebagai p_i. Dalam masalah yang melibatkan 2 kelas, distribusi kelas pada suatu *node* dapat ditulis sebagai (p_0, p_1), dimana $p_1 = 1$

$- p_0$. Sebagai ilustrasi, perhatikan kondisi tes yang ditunjukkan dalam Gambar 4.12. Distribusi kelas sebelum pemsahan adalah (0.5, 0.5) karena jumlah *record* untuk setiap kelas adalah sama. Jika kita membagi data dengan menggunakan atribut `Gender`, maka distribusi kelas dari *child* berturut-turut adalah (0.6, 0.4) dan (0.4, 0.6). Dalam pemisahan tersebut, *child* masih mengandung *record* dari kedua kelas. Pemisahan kedua menggunakan atribut `Car Type`.

Ukuran yang dibuat untuk menyeleksi pemisahan yang paling baik seringkali berdasarkan pada derajat kekotoran (*impurity*) dari *child*. Semakin kecil derajat *impurity*, maka distribusi kelas semakin tidak simetris. Sebagai contoh, sebuah *node* dengan distribusi kelas (0,1) memiliki derajat *impurity* sama dengan 0, sedangkan *node* dengan distribusi kelas seragam (0.5, 0.5) memiliki derajat *impurity* yang paling tinggi. Contoh-contoh ukuran *impurity* meliputi:

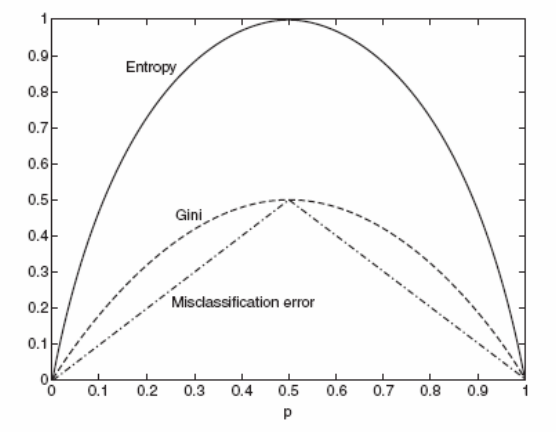
$$\textit{Entropy}(t) = -\sum_{i=0}^{c-1} p(i \mid t) \log_2 p(i \mid t) \tag{4.3}$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} \left[p(i \mid t)\right]^2 \tag{4.4}$$

$$\text{Classification } \textit{Error}(t) = 1 - \max_i \left[p(i \mid t)\right] \tag{4.5}$$

dimana c adalah banyaknya kelas dan $0\log_2 0 = 0$ dalam kalkulasi *entropy*.

Gambar 4.13 membandingkan nilai-nilai dari ukuran-ukuran *impurity* untuk masalah klasifikasi biner. p menyatakan fraksi dari *record* yang merupakan anggota salah satu dari dua kelas. Perhatikan bahwa semua ukuran mencapai nilai maksimumnya ketika distribusi kelas adalah seragam (yaitu ketika p = 0.5). Nilai ukuran minimum dicapai ketika semua *record* merupakan anggota dari kelas yang sama (yaitu ketika p sama dengan 0 atau 1). Berikut ini adalah contoh-contoh perhitungan ukuran-ukuran *impurity*.



Gambar 4.13 Perbandingan antara ukuran-ukuran *impurity* untuk masalah klasifikasi biner.

Node N_1	Count
Class = 0	0
Class = 1	6

Gini = $1 - (0/6)^2 - (6/6)^2 = 0$
Entropy = $-(0/6)\log_2(0/6) - (6/6)\log_2(6/6) = 0$
Error = $1 - \max[0/6, 6/6] = 0$

Node N_2	Count
Class = 0	1
Class = 1	5

Gini = $1 - (1/6)^2 - (5/6)^2 = 0.278$
Entropy = $-(1/6)\log_2(1/6) - (5/6)\log_2(5/6) = 0.650$
Error = $1 - \max[1/6, 5/6] = 0.167$

Node N_1	Count
Class = 0	3
Class = 1	3

Gini = $1 - (3/6)^2 - (3/6)^2 = 0.5$
Entropy = $-(3/6)\log_2(3/6) - (3/6)\log_2(3/6) = 1$
Error = $1 - \max[3/6, 3/6] = 0.5$

Contoh-contoh sebelumnya, sesuai dengan Gambar 4.13, mengilustrasikan kekonsistenan diantara ukuran-ukuran *impurity* yang berbeda. Berdasarkan perhitungan tersebut, *node* N_1 memiliki nilai *impurity* yang paling kecil, diikuti oleh N_2 dan N_3.

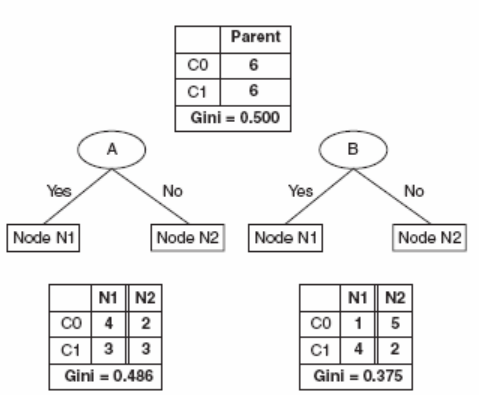
Untuk menentukan seberapa baik kondisi tes bekerja, kita perlu membandingkan derajat *impurity* dari *node parent* (sebelum pemisahan) dengan derajat *impurity* dari *node child* (setelah pemisahan). Semakin besar perbedaannya, semakin baik kondisi tes. Gain, Δ, adalah kriteria yang dapat digunakan untuk menentukan kualitas pemisahan:

$$\Delta = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \qquad (4.6)$$

dimana I(.) adalah ukuran *impurity* dari *node* yang diberikan, N adalah total banyaknya *record* pada *node parent*, k adalah banyaknya nilai atribut dan $N(v_j)$ adalah banyaknya *record* yang bersesuaian dengan *node child*, v_j. Algoritme induksi pohon keputusan seringkali memilih kondisi tes yang memaksimumkan gain, Δ. Karena I(*parent*) adalah sama untuk semua kondisi tes, memaksimumkan gain ekuivalen dengan meminimumkan ukuran *impurity* rata-rata berbobot dari *child*. Akhirnya, ketika *entropy* digunakan sebagai ukuran *impurity* dalam persamaan 4.6, perbedaan dalam *entropy* dikenal sebagai information gain, Δ_{info}.

Pemisahan Atribut Biner

Perhatikan diagram pada Gambar 4.14. Anggap terdapat dua cara untuk memisahkan data ke dalam *subset* yang lebih kecil. Sebelum pemisahan, *Gini index* adalah 0.5 karena terdapat sejumlah yang sama dari *record* dalam kedua kelas tersebut. Jika atribut A dipilih untuk memisahkan data, *Gini index* untuk *node* N1 adalah 0.4898, dan untuk N2 adalah 0.480. Rataan berbobot dari *Gini index* untuk *node* keturunan adalah $(7/12) \times 0.4898 + (5/12) \times 0.480 = 0.486$. Secara sama, kita dapat menunjukkan bahwa rataan berbobot dari *Gini index* untuk atribut B adalah 0.375. Karena *subset-subset* untuk atribut B memiliki *Gini index* yang lebih kecil, maka atribut B dipilih daripada A.



Gambar 4.14 Pemisahan atribut biner

Pemisahan Atribut Nominal

Atribut nominal dapat menghasilkan pemisahan biner atau *multiway* seperti ditunjukkan dalam Gambar 4.15. Komputasi dari *Gini index* untuk pemisahan biner mirip dengan penentuan atribut-atribut biner. Untuk pengelompokan biner pertama dari atribut `Car Type`, *Gini index* dari {`Sports`, `Luxury`} adalah 0.4922 dan *Gini index* dari {`Family`} adalah 0.3750. *Gini index* rataan berbobot untuk pengelompokan sama dengan

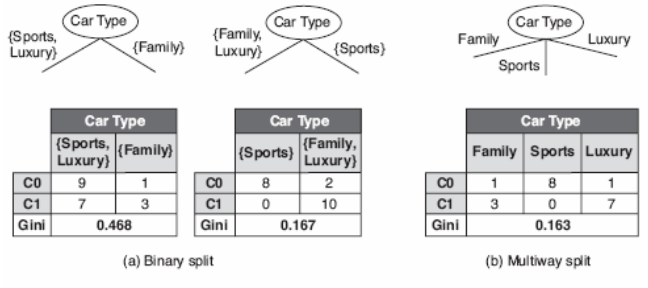
$$16/20 \times 0.4922 + 4/20 \times 0.3750 = 0.468.$$

Secara sama, pengelompokkan biner kedua dari {`Sports`} dan {`Family, Luxury`}, *Gini index* rataan berbobot adalah 0.167.

Untuk pemisahan *multiway*, *Gini index* dihitung untuk setiap nilai atribut. Karena Gini({`Family`}) = 0.375, Gini({`Sports`}) = 0 dan Gini({`Luxury`}) = 0.219, *Gini index* keseluruhan untuk pemisahan *multiway* sama dengan

$$4/20 \times 0.375 + 8/20 \times 0 + 8/20 \times 0.219 = 0.163.$$

Pemisahan *multiway* memiliki *Gini index* lebih kecil dibandingkan dengan pemisahan two-way. Hasil ini tidak mengherankan karena pemisahan two-way secara aktual menngabungkan beberapa keluaran dari pemisahan *multiway*.



Gambar 4.15 Pemisahan atribut nominal

Pemisahan Atribut Kontinu

Perhatikan contoh dalam Gambar 4.16, dimana kondisi tes `Annual Income` ≤ v digunakan untuk memisahkan *training set* untuk masalah klasifikasi kelalaian peminjaman. Metode brute-force digunakan untuk setiap nilai atribut dalam N *record* untuk menentukan v sebagai kandidat untuk posisi pemisahan. Untuk setiap kandidat v, *data set* di-scan sekali untuk menghitung banyaknya *record* dengan `Annual Income` lebih kecil atau lebih besar dari v. Kemudian *Gini index* dihitung untuk setiap kandidat dan memilih salah satu yang memberikan nilai paling kecil. Kandidat untuk posisi pemisahan diidentifikasi dengan mengambil titik tengah diantara dua nilai terurut yang saling berdekatan: 55, 65, 72 dan seterusnya. Dalam mengevaluasi *Gini index* dari kandidat posisi pemisahan tidak perlu semua N *record* dievaluasi.

Untuk kandidat pertama, v = 55, tidak ada satupun *record* yang memiliki `Annual Income` yang lebih kecil dari $55K. Hasilnya, *Gini index* untuk *node* keturunan dengan `Annual Income` < $55K adalah nol. Sedangkan banyaknya *record* dengan `Annual Income` lebih besar atau sama dengan $55K adalah 3 (untuk kelas Yes) dan 7 (untuk kelas No). Dengan demikian *Gini index* untuk *node* tersebut adalah 0.420. *Gini index* keseluruhan untuk kandidat posisi pemisahan ini sama dengan $0 \times 0 + 1 \times 0.420 = 0.420$.

Untuk kandidat yang kedua, v = 65, kita dapat menentukan distribusi kelasnya dengan memperbaharui distribusi dari kandidat sebelumnya. Distribusi baru diperoleh dengan menguji label kelas untuk *record* tersebut dengan `Annual Income` paling rendah (yaitu $60K). Karena label kelas untuk *record* ini adalah No, count untuk kelas No meningkat dari 0 ke 1 (untuk `Annual Income` ≤ $65K) dan menurun dari 7 ke 6 (untuk `Annual Income` > $65K). Distribusi untuk kelas Yes tidak berubah. Rataan berbobot dari *Gini index* yang baru untuk kandidat posisi pemisahan adalah 0.400.

Prosedur ini diulang sampai nilai *Gini index* untuk semua kandidat dihitung, seperti ditunjukkan dalam Gambar 4.16. Posisi pemisahan terbaik adalah v = 97, yang menghasilkan nilai *Gini index* paling kecil.

Class	No		No		No		Yes		Yes		Yes		No		No		No		No	
	Annual Income																			
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420	

Gambar 4.16 Pemisahan atribut kontinu

Prosedur ini memerlukan sejumlah konstan waktu untuk memperbaharui distribusi kelas pada setiap kandidat posisi pemisahan. Prosedur ini dapat dioptimisasi dengan cara menempatkan kandidat posisi pemisahan diantara dua dua *record* yang bersebelahan dengan label kelas yang berbeda. Sebagai contoh, karena tiga *record* terurut pertama (dengan `Annual Income` $60K, $70K dan $75K) memiliki label kelas yang identik, posisi pemisahan terbaik tidak menempati lokasi diantara $60K dan $75K. Dengan demikian, kandidat posisi pemisahan terbaik pada v = $55K, $65K, $72K, $87K, $92K, $110K, $122K, $172K, dan

$230K diabaikan karena nilai-nilai tersebut dilokasikan antara dua *record* yang besebelahan dengan label kelas yang sama. Pendekatan ini dapat mengurangi banyaknya kandidat posisi pemisahan dari 11 menjadi 2.

Rasio Gain

Ukuran *impurity* seperti *entropy* dan *Gini index* cenderung menyukai atribut-atribut yang memiliki sejumlah besar nilai-nilai yang berbeda. Gambar 4.12 menunjukkan tiga alternatif kondisi tes untuk mempartisi *data set* berikut:

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Dengan perbandingan kondisi tes yang pertama, `Gender`, dengan kondisi tes yang kedua, `Car Type`, dengan mudah dapat dilihat bahwa `Car Type` seperti memberikan cara terbaik dalam pemisahan data.

4.3.5 Algoritme Induksi Pohon Keputusan

Algoritme induksi pohon keputusan yang dinamakan TreeGrowth adalah

Algorithm 4.1 A skeleton decision tree induction algorithm.

```
TreeGrowth (E, F)
 1: if stopping_cond(E,F) = true then
 2:    leaf = createNode().
 3:    leaf.label = Classify(E).
 4:    return leaf.
 5: else
 6:    root = createNode().
 7:    root.test_cond = find_best_split(E, F).
 8:    let V = {v|v is a possible outcome of root.test_cond }.
 9:    for each v ∈ V do
10:       E_v = {e | root.test_cond(e) = v and e ∈ E}.
11:       child = TreeGrowth(E_v, F).
12:       add child as descendent of root and label the edge (root → child) as v.
13:    end for
14: end if
15: return root.
```

Input untuk algoritme ini adalah *record training* E dan himpunan atribut F. Algoritme tersebut bekerja secara rekursif dengan memilih atribut terbaik untuk memisahkan data (langkah 7) dan memperluas *leaf node* pada *tree* (langkah 11 dan 12) sampai kriteria berhenti dipenuhi (langkah 1). Berikut adalah penjelasan lebih detil mengenai algoritme tersebut:

1. Fungsi `createNode()` memperluas pohon keputusan dengan membuat *node* baru. Sebuah *node* dalam pohon keputusan memiliki sebuah kondisi tes, yang dinotasikan sebagai *node.test_cond*, atau label kelas, yang dinotasikan sebagai *node_label*.

2. Fungsi `find_best_split()` menentukan atribut mana yang harus dipilih sebagai kondisi tes untuk pemisahan *training record*. Seperti telah dijelaskan sebelumnya, pemilihan kondisi tes tergantung pada ukuran *impurity* yang digunakan untuk menentukan kualitas dari pemisahan. Beberapa ukuran yang digunakan adalah *entropy*, *Gini index*, dan statistik χ^2.

3. Fungsi `Classify()` menentukan label kelas untuk diberikan ke *leaf node*. Untuk setiap *leaf node*, misalkan p(i|t) menyatakan fraksi dari *training record* dari kelas I yang berhubungan dengan *node* t. Dalam banyak kasus, *leaf node* diberikan ke kelas yang memiliki banyaknya *training record* mayoritas:

$$leaf.label = \frac{\arg\max p(i \mid t)}{i}$$

   dimana operator argmax mengembalikan argumen I yang memaksimumkan pernyataan p(i|t). Selain memberikan informasi yang diperlukan untuk menentukan label kelas dari *leaf node*, fraksi p(i|t) dapat juga digunakan untuk menduga probabilitas bahwa sebuah *record* yang diberikan ke *leaf node* t anggota dari kelas i.

4. Fungsi `stopping_cond()` digunakan untuk menghentikan proses pertumbuhan *tree* dengan menguji apakah semua *record* memiliki label kelas yang sama atau nilai atribut yang sama. Cara lain untuk menghentikan fungsi rekursif adalah menguji apakah banyaknya *record* telah berada di bawah nilai *threshold* minimum tertentu.

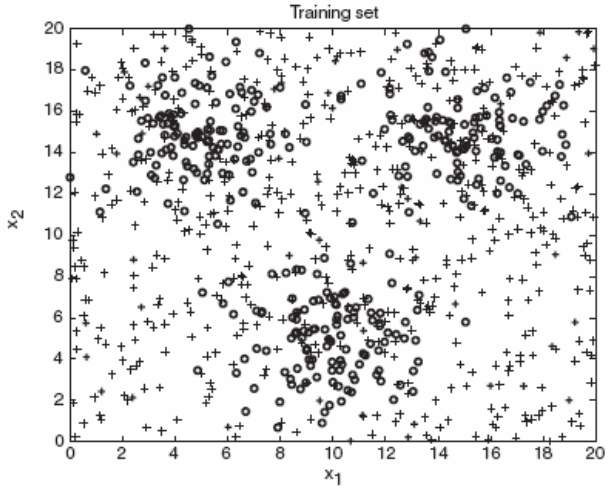
Setelah membangun pohon keputusan, langkah *tree-pruning* dapat dilakukan untuk mengurangi ukuran dari pohon keputusan. *Pruning* dilakukan dengan memangkas cabang-cabang dari pohon awal untuk menigkatkan kapabilitas generalisasi dari pohon keputusan.

4.4 Model *Overfitting*

Error yang dilakukan dari model klasifikasi secara umum dibagi ke dalam dua bentuk, yaitu *training error* dan generalization *error*. *Training error* juga dikenal sebagai resubstitution *error* atau ap*parent error*, adalah bilangan misclassification *error* yang dilakukan pada *training record*, sedangkan generalization *error* adalah expected *error* dari model pada *record-record* yang belum terlihat sebelumnya.

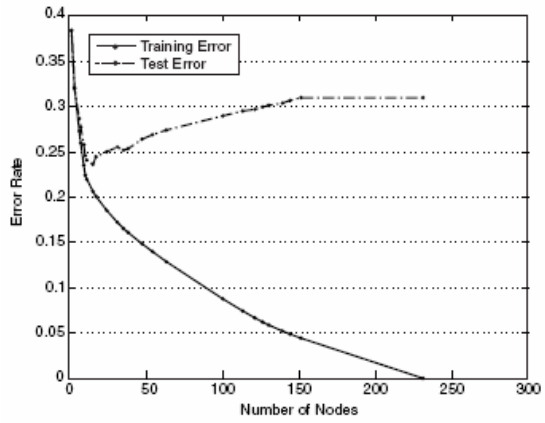
Seperti yang dijelaskan dalam sub bab sebelumnya bahwa model klasifikasi yang baik tidak hanya cocok dengan *training* data, tetapi juga harus secara akurat mengklasifikasikan *record-record* yang belum pernah dilihatnya sebelumnya. Dengan kata lain, model yang baik harus memiliki *training error* yang rendah dan juga generalization *error* yang rendah pula. Hal ini penting karena sebuah model yang cocok dengan *training* data dengan begitu baik dapat memiliki generalization yang paling buruk dibandingkan dengan model dengan *training error* yang paling tinggi. Situasi demikian dikenal sebagai model *overfitting*.

Contoh *overfitting* dalam data Dua-Dimensi. Untuk contoh nyata masalah *overfitting*, perhatikan *data set* dua-dimensi yang ditunjukkan dalam Gambar 4.17.



Gambar 4.17 Contoh *data set* dengan kelas-kelas biner

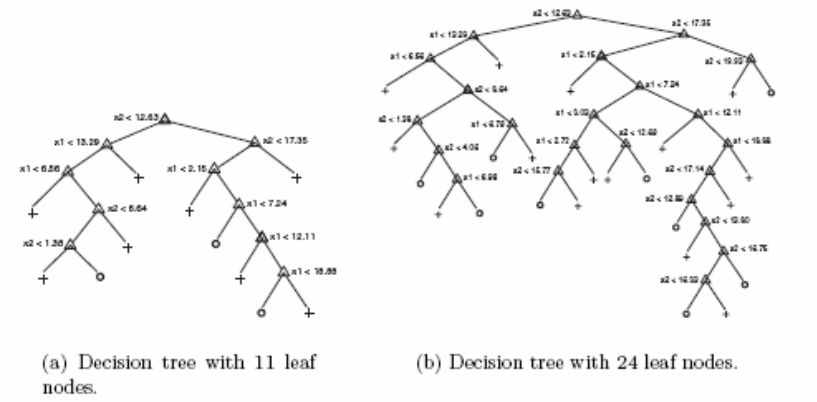
Data set mengandung titik data yang merupakan anggota dari kelas yang berbeda, yang dinotasikan sebagai kelas o dan kelas + . Titik data untuk kelas o di-generate dari campuran tiga distribusi Gaussian, sedangkan titik data untuk kelas + di-generate dengan menggunakan distribusi seragam. Terdapat 1200 titik anggota kelas o dan 1800 titik anggota kelas +. 30% dari titik-titik ersebut dipilih untuk *training*, sedangkan 70% sisanya digunakan untuk testing. *Classifier* pohon keputusan yang menggunakan *Gini index* sebagai ukuran *impurity*-nya diaplikasikan ke *training set*. Untuk menyelidiki pengaruh *overfitting*, level yang berbeda dari *pruning* diaplikasikan ke pohon awal. Gambar 4.18 menunjukkan tingkat *training error* dan test *error* dari pohon keputusan.



Gambar 4.18 Tingkat *training error* dan *test error*

Perhatikan bahwa tingkat *training error* dan *test error* dari model menjadi lebih besar ketika ukuran dari *tree* sangat kecil. Situasi ini dikenal sebagai model *underfitting*. *Underfitting* terjadi karena model masih mempelajari struktur dari data. Hasilnya, *tree* bekerja dengan buruk pada *training* dan *test set*. Sebagaimana banyaknya *node* dalam pohon keputusan meningkat, *tree* memiliki *training error* dan test *error* yang lebih kecil. Pada saat *tree* berukuran sangat besar, tingkat test *error*-nya mulai meningkat walaupun tingkat *training error*-nya terus menurun. Fenomena ini dikenal sebagai model *overfitting*.

Untuk memahami fenomena *overfitting*, perhatikan bahwa *training error* dari model dapat dikurangi dengan meningkatkan kompleksitas dari modelnya. Sebagai contoh, *leaf node* dari *tree* dapat diperluas sampai betul-betul sesuai dengan *training* data. Walaupun *training error* untuk *tree* demikian adalah nol, test *error* dapat menjadi besar karena *tree* dapat mengandung *node-node* yang secara tidak sengaja sesuai dengan beberapa titik *noise* dalam *training* data. *Node* demikian dapat menurunkan kinerja dari *tree* karena *node-node* tersebut tidak men-generalisasi dengan baik contoh-contoh test. Gambar 4.19 menunjukkan struktur dari dua pohon keputusan dengan banyaknya *node* yang berbeda. *Tree* yang mengandung sejumlah kecil *node* memiliki tingkat *training error* yang lebih tinggi, tetapi memiliki tingkat test *error* yang lebih rendah dibandingkan dengan *tree* yang lebih kompleks.



(a) Decision tree with 11 leaf nodes.

(b) Decision tree with 24 leaf nodes.

Gambar 4.19 *Decision tree* dengan kompleksitas model yang berbeda

4.4.1 *Overfitting* Disebabkan karena Adanya *Noise*

Perhatikan *training* dan *test set* dalam Tabel 4.3 dan 4.4 untuk masalah klasifikasi mammal. Dua dari 10 *training record* adalah mislabeled, yaitu bats dan whales yang diklasifikasikan sebagai non-mammal bukan sebagai mammal.

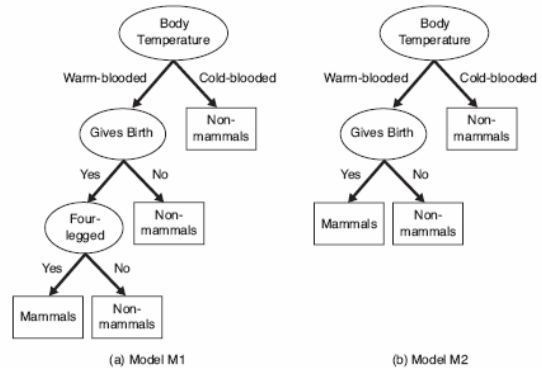
Tabel 4.3 Contoh *training set* untuk mengklasifikasi mammal. Label kelas dengan simbol bintang menyatakan *record* yang salah dalam pemberian label (mislabeled).

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Tabel 4.4 Contoh *test set* untuk mengklasifikasi mammal.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	cold-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

Sebuah pohon keputusan yang secara sempurna sesuai (cocok) dengan *training set* ditunjukkan dalam Gambar 4.20(a). Walapun *training error* untuk *tree* adalah nol, tingkat *error*-nya pada *test set* adalah 30%. Terjadi kesalahan klasifikasi pada human maupun dolphin. Keduanya diklasifikasikan sebagai non-mammal karena nilai atribut untuk `Body Temperature`, `Give Birth`, dan `Four-legged` identik dengan *record* yang mislabeled dalam *training set*.



Gambar 4.20 Pohon keputusan yang dihasilkan dari *data set* yang ditunjukkan dalam Tabel 4.3

Di lain pihak spiny anteaters mereprsentasikan kasus eksepsional dimana label kelas dari test *record* kontradiksi dengan label kelas dari *record* yang serupa dalam *training set*. *Error* yang disebabkan oleh kasus eksepsional sering kali tidak dapat dihindari dan membentuk tingkat *error* minimum yang dapat dicapai oleh *classifier*.

Sebaliknya pohon keputusan M2 yang ditunjukkan dalam Gambar 4.20(b) memiliki tingkat test *error* yang paling rendah (10%) walaupun tingkat *training error*-nya cukup tinggi (20%). Jelaslah bahwa pohon keputusan pertama, M1, telah meng-*overfit training* data karena terdapat model yang lebih sederhana dengan tingkat *error* yang paling rendah pada *test test*. Kondisi test atribut `Four-legged` dalam model M1 adalah palsu karena kondisi tersebut mencocokan mislabeled *training record*, yang mengakibatkan misclassification dari *record-record* dalam *test set*.

4.4.2 *Overfitting* Disebabkan karena Kurangnya Sample-sample yang Representatif

Model yang membuat keputusan klasifikasinya berdasarkan pada sejumlah kecil *training record* juga mudah terkena *overfitting*. Model demikian dapat

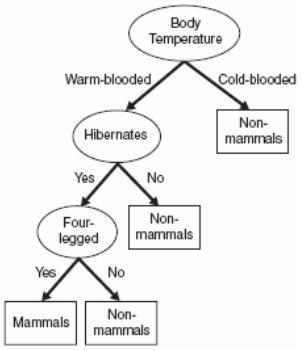
dibangun karena kurangnya sample-sample representatif dalam *training* data dan algortime pembelajaran yang terus memperbaiki modelnya walaupun ketika sedikit *training record* yang tersedia. Sebagai ilustrasi perhatikan contoh berikut.

Perhatikan lima *training record* seperti yang ditunjukkan dalam Tabel 4.5. Semua dari *training record* diberi label secara benar dan pohon keputusan yang terkait digambarkan dalam Gambar 4.21. Walaupun *training error* adalah nol, tetapi tingkat *error* pada *test set* adalah 30%.

Tabel 4.5. Contoh *training set* untuk mengklasifikasi mammal.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
salamander	cold-blooded	no	yes	yes	no
guppy	cold-blooded	yes	no	no	no
eagle	warm-blooded	no	no	no	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes

Pada human, elephant, dan dolphin salah diklasifikasikan karena pohon keputusan mengklasifikasikan semua *warm-blooded vertebrate* yang tidak tidur pada musim dingin sebagai non-mammal. Pohon mencapai keputusan klasifikasi ini karena hanya terdapat satu *record training*, yaitu eagle, dengan karakteristik demikian. Contoh ini menggambarkan pembuatan keputusan yang salah ketika tidak terdapat contoh representatif yang cukup pada *leaf node* dari sebuah pohon keputusan.



Gambar 4.21 Pohon keputusan yang dihasilkan dari *data set* yang ditunjukkan dalam Tabel 4.5.

4.4.3 Mengestimasi *Error* Generalisasi

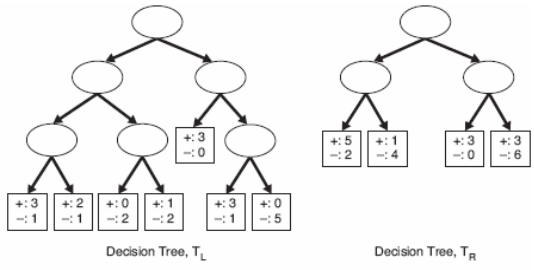
Dalam masalah klasifikasi salah satu isu penting adalah bagaimana menentukan kompleksitas model yang benar. Kompleksitas ideal terjadi pada model yang menghasilkan *error* generalisasi yang paling rendah. Masalah yang ditemui adalah algoritme pembalajaran hanya memiliki akses ke *training set*, dan tidak memiliki pengetahuan terhadap *test set*. Dengan demikian, algoritme tidak mengetahui apakah *tree* akan bekerja dengan baik pada *record* yang belum pernah dilihatnya sebalumnya. Algoritme yang baik harus dapat mengestimasi *error* generalisasi dari pohon dihasilkan.

Menggunakan Estimasi Resubtitusi

Pendekatan estimasi resubtitusi mengasumsikan bahwa *training set* adalah representasi yang baik dari data keseluruhan. Dengan demikian *training error*

dapat digunakan untuk memberikan estimasi yang optimis untuk *error* generalisasi. Dengan asumsi ini, algoritme induksi pohon keputusan memilih model yang menghasilkan tingkat *training error* yang paling kecil dari *error* generalisasi.

Contoh 4.1. Perhatikan pohon keputusan biner yang ditunjukkan dalam Gambar 4.22. Diasumsikan bahwa kedua pohon keputusan tersebut dibangun dari *training data* yang sama dan keduanya membuat keputusan-keputusan klasifikasinya pada setiap *leaf node* berdasarkan kelas majority. Perhatikan bahwa pohon bagian kiri, T_L, lebih kompleks karena mengekspansi beberapa *leaf node* dalam pohon bagian kanan, T_R. Tingkat *training error* untuk pohon pada bagian kiri adalah $e(T_L) = 4/24 = 0.167$, sedangkan *training error* untuk pohon pada bagian kanan adalah $e(T_R) = 6/24 = 0.25$. Berdasarkan pada *error* resubstitusinya, pohon pada bagian kiri dipandang lebih baik dibandingkan dengan pohon pada bagian kanan.



Gambar 4.22 Contoh dua pohon keputusan yang di-*generate* dari *training data* yang sama.

4.4.4 Menangani *Overfitting* dalam Induksi Pohon Keputusan

Terdapat dua strategi untuk menghindari model *overfitting* dalam komteks induksi pohon keputusan.

Prepruning. Dalam pendekatan ini, algoritme yang sedang membentuk pohon dihentikan sebelum men-generate pertumbuhan pohon secara penuh yang secara sempurna mencocokan seluruh *training* data. Untuk keperluan tersebut, kondisi berhenti digunakan; sebagai contoh berhenti meng-ekspansi *leaf node* ketika gain yang diobservasi (atau peningkatan dalam *error* generalisasi yang diestimasi) dalam ukuran *impurity* berada di bawah nilai *threshold* tertentu. Keuntungan dari pendekatan ini adalah dapat dihindarinya pembangunan sub*tree* yang kompleks yang meng-*overfit training* data. Pemilihan nilai *threshold* yang benar untuk menghentikan algoritme adalah cukup sulit. Nilai *threshold* yang terlalu tinggi dapat menghasilkan *underfitted model*, sedangkan nilai *threshold* yang terlalu rendah mungkin kurang cukup untuk menyelesaikan masalah *model overfiting*.

Post-pruning. Dalam pendekatan ini, pohon keputusan dibangun sampai ukuran maksimumnya. Pendekatan ini diikuti oleh sebuah langkah *tree-pruning* yang memangkas pohon secara bottom-up. Pemangkasan dapat dilakukan dengan menghapuskan sebuah sub*tree* dengan (1) sebuah *leaf node* baru yang memiliki label kelas ditentukan dari kelas mayoritas dari *record* yang digabungkan dengan sub*tree*, atau (2) cabang dari sub*tree* yang sering digunakan. Langkah *tree-pruning* berakhir ketika tidak ada lagi peningkatan yang diperoleh. Post-*pruning* cenderung memberikan hasil yang lebih baik dari pre*pruning* karena post-*pruning* membuat keputusan *pruning* berdasarkan pada keseluruhan pohon, tidak seperti

pre*pruning*, yang dapat mengalami penghentian proses pembentukan pohon secara prematur. Walaupun demikian, *port-pruning* tambahan komputasi yang diperlukan untuk membentuk pohon secara keseluruhan dapat menjadi pemborosan ketika sub*tree* dipangkas.

4.5 Evaluasi Kinerja *Classifier*

Ukuran kinerja dari model pada *test set* sering kali berguna karena ukuran tersebut memberikan estimasi yang tidak bias dari *error* generalisasinya. Akurasi dari tingkat *error* yang dihitung dari *test set* dapat juga digunakan untuk membandingkan kinerja relatif dari *classifier-classifier* pada domain yang sama. Untuk melakukan perbandingkan ini, label kelas dari test *record* haruslah diketahui. Berikut adalah metode yang umum digunakan untuk mengevaluasi kinerja *classifier*.

4.5.1 Holdout Method

Dalam holdout method, data awal dengan contoh-contoh yang diberi label dipartisi ke dalam dua himpunan disjoint, dimanakan *training set* dan *test set*. Model klasifikasi selanjutnya dihasilkan dari *training set* dan kinerjanya dievaluasi pada *test set*. Proporsi data yang dicadangkan untuk *training set* dan *test set* tergantung pada analisis (contoh 50-50, atau 2/3 untuk *training* dan 1/3 untuk testing). Akurasi dari *classifier* dapat diestimasi berdasarkan pada akurasi model yang dihasilkan pada *test set*.

Holdout method memiliki beberapa keterbatasan. Pertama, lebih sedikit contoh-contoh berlabel yang tersedia untuk *training* karena beberapa *record* digunakan untuk testing. Hasilnya adalah model yang dihasilkan dapat tidak sebagus ketika semua contoh berlabel digunakan untuk *training*. Kedua, model dapat sangat tergantung pada komposisi dari *training set* dan *test set*. Semakin kecil ukuran *training set*, semakin besar variance dari model. Di lain pihak, jika *training set* terlalu besar, maka akurasi yang diestimasi yang dihitung dari *test set* yang lebih kecil adalah kurang reliable. Estimasi demikian dikatakan memiliki selang kepercayaan yang lebar. Akhirnya *training set* dan *test set* tidak lagi tergantung satu dengan lainnya.

4.5.2 Random Subsampling

Metode Holdout dapat diulangi beberapa kali untuk meningkatkan estimasi dari kinerja *classifier*. Pendekatan ini dikenal sebagai *random subsampling*. Misalkan acc_i adalah akurasi model selama iterasi ke-i. Akurasi keseluruhan diberikan oleh $acc_{sub} = \sum_{i=1}^{k} acc_i / k$. Random sampling masih menjumpai beberapa masalah yang terkait dengan metode Holdout karena tidak menggunakan sebanyak mungkin data untuk *training*. Metode ini juga tidak memiliki kontrol terhadap berapa kali setiap *record* digunakan untuk *training* dan testing. Dengan demikian, beberapa *record* dapat digunakan untuk *training* lebih sering dibanding dengan *record-record* yang lain.

4.5.3 *Cross-Validation*

Dalam pendekatan *cross-validation*, setiap *record* digunakan beberapa kali dalam jumlah yang sama untuk *training* dan tepat sekali untuk testing. Untuk mengilustrasikan metode ini, anggaplah kita mempartisi data ke dalam dua *subset* yang berukuran sama. Pertaman, kita pilih satu dari kedua *subset* tersebut untuk *training* dan satu lagi untuk testing. Kemudian dilakukan pertukaran fungsi dari *subset* sedemikian sehingga *subset* yang sebelumnya sebagai *training set* menjadi *test set* demikian sebaliknya. Pendekatan ini dinamakan two-fold cross-validation. Total *error* diperoleh dengan menjumlahkan *error-error* untuk kedu proses tersebut. Dalam contoh ini, setiap *record* digunakan tepat satu kali untuk *training* dan satu kali untuk testing. Metode *k-fold cross-validation* men-generalisasi pendekatan ini dengan mensegmentasi data ke dalam k partisi berukuran sama. Selama proses, salah satu dari partisi dipilih untuk testing, sedangkan sisanya digunakan untuk *training*. Prosedur ini diulangi k kali sedemikian sehingga setiap partisi digunakan untuk testing tepat satu kali. Total *error* ditentukan dengan menjumlahkan *error* untuk semua k proses tersebut. Kasus khusus untuk metode *k-fold cross-validation* menetapkan k = N, ukuran dari *data set*. Metode ini dinamakan pendekatan *leave-one-out*, setiap *test set* hanya mengandung satu *record*. Pendekatan ini memiliki keuntungan dalam pengunaan sebanyak mungkin data untuk *training*. *Test set* bersifat mutually exclusive dan secara efektif mencakup keseluruhan *data set*. Kekurangan dari pendekatan ini adalah banyaknya komputasi untuk mengulangi prosedur sebanyak N kali.

4.5.4 *Bootstrap*

Metode yang telah dijelaskan sebelumnya mengasumsikan bahwa *training record* dinyatakan sebagai sample tanpa pergantian. Hasilnya adalah tidak ada *error* duplikat dalam *training* dan *test set*. Dalam pendekatan bootstrap, *training record* dinyatakan sebagai sample dengan pergantian; yaitu *record* yang telah dipilih untuk *training* ditempatkan kembali ke dalam tempat penyimpanan *record* awal sedemikian sehingga *record* tersebut memiliki peluang yang sama untuk dipilih kembali. Jika data awla memiliki N *record*, dapat ditunjukkan bahwa secara rata-rata, bootstrap sample dengan ukuran N mengandung 63.2% *record* dalam data awal. Aproksimasi ini mengikuti fakta bahwa probabilitas sebuah *record* dipilih oleh sebuah bootstrap adalah $1 - (1 - 1/N)^N$. Ketika N cukup besar, probabilitas secara asimtotik mendekati $1 - e^{-1} = 0.632$. *Record* yang tidak termasuk dalam *bootstrap sample* menjadi bagian dari *test set*. Model yang dihasilkan dari *training set* kemudian diaplikasikan ke *test set* untuk mendapatkan estimasi dari akurasi *bootstrap sample*, ε_i. Prosedur sampling ini kemudian diulangi b kali untuk men-*generate* b *bootstrap sample*.

Terdapat beberapa variasi untuk pendekatan bootstrap sampling dalam hal perhitungan akurasi keseluruhan dari *classifier*. Salah satu pendekatan yang paling banyak digunakan adalah .632 bootstrap, yang menghitung akurasi keseluruhan dengan mengkombinasikan akurasi dari setiap *bootstrap sample*, ε_i, dengan akurasi yang dihitung dari traning set yang mengandung semua contoh-contoh berlabel dalam data awal (acc_s):

$$\text{Accuracy, } acc_{boot} = \frac{1}{b}\sum_{i=1}^{b}(0.632 \times \varepsilon_i + 0.368 \times \varepsilon_s)$$

Penutup – Soal Latihan

Tugas Individu

Jawablah pertanyaan berikut secara singkat dan jelas.

1. Diberikan *training example* untuk masalah klasifikasi biner berikut

Customer ID	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

   a. Hitunglah Gini *index* untuk seluruh koleksi *training example*.

   b. Hitunglah Gini *index* untuk atribut `Customer ID`

   c. Hitunglah Gini *index* untuk atribut `Gender`

   d. Hitunglah Gini *index* untuk atribut `Car Type` dengan menggunakan *multiway split*.

   e. Hitunglah Gini *index* untuk atribut `Shirt Size` dengan menggunakan *multiway split*.

   f. Yang manakah atribut yang paling baik, `Customer ID`, `Gender`, `Car Type`, dan `Shirt Size`?

2. Perhatikan *training example* berikut untuk masalah klasifikasi biner:

Instance	a_1	a_2	a_3	Target Class
1	T	T	1.0	+
2	T	T	6.0	+
3	T	F	5.0	−
4	F	F	4.0	+
5	F	T	7.0	−
6	F	T	3.0	−
7	F	F	8.0	−
8	T	F	7.0	+
9	F	T	5.0	−

   a. Tentukan *entropy* dari koleksi *training example* ini terhadap kelas positif.

   b. Tentukan *information gain* dari a1 dan a2 relatif terhadap *training example*.

c. Untuk a3, yang merupakan atribut kontinu, hitung *information gain* untuk setiap kemungkinan pemisahan.

d. Tentukan pemisahan yang terbaik (diantara a1, a2 dan a3) berdasarkan *information gain*.

e. Tentukan pemisahan yang terbaik (diantara a1, a2 dan a3) berdasarkan tingkat *error* klasifikasi.

f. Tentukan pemisahan yang terbaik (diantara a1, a2 dan a3) berdasarkan *Gini index*.