



# **Pembelajaran Mesin (Machine Learning)**





# Pembelajaran Mesin

## (Machine Learning)

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

### BIODATA PENULIS



Dr. Budi Raharjo, S.Kom, M.Kom, MM lahir di Semarang, tanggal 22 Februari 1985. Beliau adalah Alumni dari Universitas Bina Nusantara (BINUS University) Jakarta dan juga alumni Universitas Kristen Satya wacana (UKSW) Salatiga. Dr. Budi Raharjo telah menjadi Dosen pada Universitas STEKOM pada mata kuliah Kepemimpinan (Leadership), mata kuliah Pengantar Akuntansi, Manajemen Proses, Manajemen Akuntansi dan Manajemen Resiko Bisnis. Selain sebagai dosen Universitas STEKOM, Dr. Budi Raharjo, M.Kom, MM juga mempunyai bisnis sendiri dalam bidang perhotelan dan juga sebagai wirausaha dalam bidang pemasok unggas (ayam) beku, ke berbagai kota besar, khususnya Jakarta dan sekitarnya.

Pengalaman beliau berwirausaha menjadi bekal utama dalam penulisan buku ajar yang diterbitkan oleh Yayasan Prima Agus Teknik (YPAT) Semarang. Oleh sebab itu bukunya berisi langkah-langkah praktis yang mudah diikuti oleh para mahasiswa, saat mahasiswa mengikuti proses perkuliahan pada Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Jabatan struktural yang diembannya saat ini adalah Wakil Rektor 1 (Akademik) Universitas STEKOM Semarang.



YAYASAN PRIMA AGUS TEKNIK

**YAYASAN PRIMA AGUS TEKNIK**  
Jl. Majapahit No. 605 Semarang  
Telp. (024) 6723456. Fax. 024-6710144  
Email : [penerbit\\_ypat@stekom.ac.id](mailto:penerbit_ypat@stekom.ac.id)

ISBN 978-623-5734-23-1



# **Pembelajaran Mesin (Machine Learning)**

**Dr. Budi Raharjo, S.Kom., M.Kom., MM.**



**YAYASAN PRIMA AGUS TEKNIK**

# **PEMBELAJARAN MESIN ( Machine Learning )**

## **Penulis :**

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

**ISBN : 9 786235 734231**

## **Editor :**

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

## **Penyunting :**

Dr. Joseph Teguh Santoso, M.Kom.

## **Desain Sampul dan Tata Letak :**

Irdha Yunianto, S.Ds., M.Kom

## **Penerbit :**

Yayasan Prima Agus Teknik

## **Redaksi :**

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : [penerbit\\_ypat@stekom.ac.id](mailto:penerbit_ypat@stekom.ac.id)

## **Distributor Tunggal :**

### **Universitas STEKOM**

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : [info@stekom.ac.id](mailto:info@stekom.ac.id)

Hak cipta dilindungi undang-undang

Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara apapun tanpa ijin dari penulis

## KATA PENGANTAR

Puji syukur pada Tuhan Yang Maha Esa bahwa buku yang berjudul “Pembelajaran Mesin (*Machine Learning*)” ini dapat diselesaikan dengan baik. Pembelajaran mesin adalah bagian integral dari banyak aplikasi komersial dan proyek penelitian saat ini, di berbagai bidang mulai dari diagnosis dan perawatan medis hingga menemukan teman Anda di jejaring sosial. Banyak orang berpikir bahwa pembelajaran mesin hanya dapat diterapkan oleh perusahaan besar dengan tim riset yang luas. Dalam buku ini, kami ingin menunjukkan kepada Anda betapa mudahnya membangun solusi pembelajaran mesin sendiri, dan cara terbaik untuk melakukannya. Dengan pengetahuan dalam buku ini, Anda dapat membangun sistem Anda sendiri untuk mengetahui bagaimana perasaan orang-orang di Twitter, atau membuat prediksi tentang pemanasan global. Aplikasi pembelajaran mesin tidak terbatas dan, dengan jumlah data yang tersedia saat ini, sebagian besar dibatasi oleh imajinasi Anda.

Buku ini disusun sebagai berikut Bab 1 memperkenalkan konsep dasar *machine learning* dan aplikasinya, serta menjelaskan persiapan yang akan digunakan di seluruh buku ini. Bab 2 dan 3 menjelaskan algoritma pembelajaran mesin aktual yang paling banyak digunakan dalam praktik, dan membahas kelebihan dan kekurangannya. Bab 4 membahas pentingnya cara merepresentasikan data yang diproses oleh *machine learning*, dan aspek data apa yang harus diperhatikan. Bab 5 mencakup metode lanjutan untuk evaluasi model dan penetapan parameter, dengan fokus khusus pada validasi silang (*Cross Validation*) dan pencarian grid. Bab 6 menjelaskan konsep *pipeline* untuk model chaining. Bab 7 menunjukkan bagaimana menerapkan metode yang dijelaskan dalam bab sebelumnya ke data teks, dan memperkenalkan beberapa teknik pemrosesan khusus teks. Bab 8 menawarkan ikhtisar tingkat tinggi, dan mencakup referensi ke topik yang lebih lanjut.

Penulis berharap akan meyakinkan Anda tentang kegunaan pembelajaran mesin di berbagai aplikasi, dan betapa mudahnya pembelajaran mesin dapat diimplementasikan dalam kehidupan sehari-hari. Banyak model yang penulis diskusikan menggunakan prinsip-prinsip dari teori probabilitas, aljabar linier, dan optimasi. Meskipun tidak perlu memahami semua detail tentang bagaimana algoritma ini diimplementasikan, penulis pikir mengetahui beberapa teori di balik algoritma akan membuat Anda menjadi ilmuwan data yang lebih baik.

Ada banyak buku bagus yang ditulis tentang teori pembelajaran mesin, dan jika penulis dapat meningkatkan minat Pembaca tentang kemungkinan membangun *machine learning*, kami sarankan Anda mengambil setidaknya satu dari mereka dan menggali lebih dalam. Akhir kata semoga buku ini bermanfaat bagi para pembaca.

Ungaran, Desember 2021

Penulis

Dr. Budi Raharjo, S.Kom. M.Kom, MM

## KATA PENGANTAR

Puji syukur pada Tuhan Yang Maha Esa bahwa buku yang berjudul “Pembelajaran Mesin (*Machine Learning*)” ini dapat diselesaikan dengan baik. Pembelajaran mesin adalah bagian integral dari banyak aplikasi komersial dan proyek penelitian saat ini, di berbagai bidang mulai dari diagnosis dan perawatan medis hingga menemukan teman Anda di jejaring sosial. Banyak orang berpikir bahwa pembelajaran mesin hanya dapat diterapkan oleh perusahaan besar dengan tim riset yang luas. Dalam buku ini, kami ingin menunjukkan kepada Anda betapa mudahnya membangun solusi pembelajaran mesin sendiri, dan cara terbaik untuk melakukannya. Dengan pengetahuan dalam buku ini, Anda dapat membangun sistem Anda sendiri untuk mengetahui bagaimana perasaan orang-orang di Twitter, atau membuat prediksi tentang pemanasan global. Aplikasi pembelajaran mesin tidak terbatas dan, dengan jumlah data yang tersedia saat ini, sebagian besar dibatasi oleh imajinasi Anda.

Buku ini disusun sebagai berikut Bab 1 memperkenalkan konsep dasar *machine learning* dan aplikasinya, serta menjelaskan persiapan yang akan digunakan di seluruh buku ini. Bab 2 dan 3 menjelaskan algoritma pembelajaran mesin aktual yang paling banyak digunakan dalam praktik, dan membahas kelebihan dan kekurangannya. Bab 4 membahas pentingnya cara merepresentasikan data yang diproses oleh *machine learning*, dan aspek data apa yang harus diperhatikan. Bab 5 mencakup metode lanjutan untuk evaluasi model dan penetapan parameter, dengan fokus khusus pada validasi silang (*Cross Validation*) dan pencarian grid. Bab 6 menjelaskan konsep *pipeline* untuk model chaining. Bab 7 menunjukkan bagaimana menerapkan metode yang dijelaskan dalam bab sebelumnya ke data teks, dan memperkenalkan beberapa teknik pemrosesan khusus teks. Bab 8 menawarkan ikhtisar tingkat tinggi, dan mencakup referensi ke topik yang lebih lanjut.

Penulis berharap akan meyakinkan Anda tentang kegunaan pembelajaran mesin di berbagai aplikasi, dan betapa mudahnya pembelajaran mesin dapat diimplementasikan dalam kehidupan sehari-hari. Banyak model yang penulis diskusikan menggunakan prinsip-prinsip dari teori probabilitas, aljabar linier, dan optimasi. Meskipun tidak perlu memahami semua detail tentang bagaimana algoritma ini diimplementasikan, penulis pikir mengetahui beberapa teori di balik algoritma akan membuat Anda menjadi ilmuwan data yang lebih baik.

Ada banyak buku bagus yang ditulis tentang teori pembelajaran mesin, dan jika penulis dapat meningkatkan minat Pembaca tentang kemungkinan membangun *machine learning*, kami sarankan Anda mengambil setidaknya satu dari mereka dan menggali lebih dalam. Akhir kata semoga buku ini bermanfaat bagi para pembaca.

Ungaran, Desember 2021  
Penulis

Dr. Budi Raharjo, S.Kom. M.Kom, MM

## DAFTAR ISI

<b>HALAMAN JUDUL .....</b>	<b>i</b>
<b>KATA PENGANTAR .....</b>	<b>iii</b>
<b>DAFTAR ISI .....</b>	<b>v</b>
<b>BAB I PENGANTAR .....</b>	<b>1</b>
1.1. Mengapa Pembelajaran Mesin? .....	1
1.2. Masalah yang Dapat Dipecahkan oleh Pembelajaran Mesin .....	2
1.3. Mengetahui Tugas Anda dan Mengetahui Data Anda .....	4
1.4. Mengapa Python? .....	5
1.5. <i>Scikit-Learn</i> .....	5
1.6. Perpustakaan dan Alat Penting .....	6
1.7. Versi yang Digunakan dalam Buku ini .....	11
1.8. Aplikasi Pertama: Mengklasifikasikan Spesies Iris .....	12
1.9. Mengukur Keberhasilan: Data Pelatihan dan Pengujian .....	15
1.10. Hal Penting: Lihat Data Anda .....	17
1.11. Membangun Model Pertama Anda: <i>k-Nearest Neighbors</i> .....	18
1.12. Membuat Prediksi .....	19
1.13. Mengevaluasi Model .....	20
1.14. Ringkasan dan Pandangan .....	21
<b>BAB 2 PEMBELAJARAN TERAWASI .....</b>	<b>23</b>
2.1 Klasifikasi dan Regresi .....	23
2.2 Generalisasi, <i>Overfitting</i> , dan <i>Underfitting</i> .....	24
2.3 Hubungan Kompleksitas Model dengan Ukuran Dataset .....	26
2.4 Algoritma Pembelajaran Mesin .....	27
2.5 Beberapa Contoh <i>Dataset</i> .....	27
2.6 <i>k-Nearest Neighbors</i> .....	31
2.7 Model Linier .....	38
2.8 Pengklasifikasi Naive Bayes .....	58
2.9 Pohon Keputusan .....	60
2.10 Mesin Vektor Dukungan Kernelized .....	79
2.11 Jaringan <i>Neural</i> (Pembelajaran Mendalam) .....	88
2.12 Perkiraan Ketidakpastian dari Pengklasifikasi .....	100
2.13 Fungsi Keputusan .....	101
2.14 Memprediksi Probabilitas .....	103
2.15 Ketidakpastian dalam Klasifikasi Multiklasi .....	105
2.16 Ringkasan dan Pandangan .....	107
<b>BAB 3 PEMBELAJARAN DAN PREPROCESSING TANPA PENGAWASAN .....</b>	<b>110</b>
3.1 Jenis Pembelajaran Tanpa Pengawasan .....	110

## BAB 1

### PENGANTAR

Pembelajaran mesin adalah tentang mengekstraksi pengetahuan dari data. Ini adalah bidang penelitian di persimpangan statistik, kecerdasan buatan, dan ilmu komputer dan juga dikenal sebagai analitik prediktif atau pembelajaran statistik. Penerapan metode pembelajaran mesin dalam beberapa tahun terakhir telah ada di mana-mana dalam kehidupan sehari-hari. Dari rekomendasi otomatis tentang film mana yang harus ditonton, hingga makanan apa yang dipesan atau produk mana yang akan dibeli, hingga radio online yang dipersonalisasi dan mengenali teman-teman Anda di foto Anda, banyak situs web dan perangkat modern memiliki algoritme pembelajaran mesin sebagai intinya. Saat Anda melihat situs web yang kompleks seperti Facebook, Amazon, atau Netflix, kemungkinan besar setiap bagian situs berisi beberapa model pembelajaran mesin.

Di luar aplikasi komersial, pembelajaran mesin memiliki pengaruh luar biasa pada cara penelitian berbasis data dilakukan saat ini. Alat-alat yang diperkenalkan dalam buku ini telah diterapkan pada beragam masalah ilmiah seperti memahami bintang, menemukan planet yang jauh, menemukan partikel baru, menganalisis urutan DNA, dan menyediakan perawatan kanker yang dipersonalisasi.

Aplikasi Anda tidak perlu berskala besar atau mengubah dunia seperti contoh-contoh ini untuk mendapatkan manfaat dari pembelajaran mesin. Dalam bab ini, kami akan menjelaskan mengapa pembelajaran mesin menjadi begitu populer dan membahas jenis masalah apa yang dapat diselesaikan menggunakan pembelajaran mesin. Kemudian, kami akan menunjukkan kepada Anda cara membuat model pembelajaran mesin pertama Anda, dengan memperkenalkan konsep-konsep penting di sepanjang jalan.

#### 1.1 MENGAPA PEMBELAJARAN MESIN?

Pada hari-hari awal aplikasi "cerdas", banyak sistem menggunakan aturan kode tangan dari keputusan "jika" dan "lain" untuk memproses data atau menyesuaikan dengan input pengguna. Pikirkan filter spam yang tugasnya adalah memindahkan pesan email masuk yang sesuai ke folder spam. Anda dapat membuat daftar hitam kata-kata yang akan mengakibatkan email ditandai sebagai spam. Ini akan menjadi contoh penggunaan sistem aturan yang dirancang ahli untuk merancang aplikasi "cerdas". Pembuatan aturan keputusan secara manual dapat dilakukan untuk beberapa aplikasi, terutama di mana manusia memiliki pemahaman yang baik tentang proses untuk dimodelkan. Namun, menggunakan aturan kode tangan untuk membuat keputusan memiliki dua kelemahan utama:

- Logika yang diperlukan untuk membuat keputusan khusus untuk satu domain dan tugas. Mengubah tugas bahkan sedikit mungkin memerlukan penulisan ulang seluruh sistem.
- Merancang aturan membutuhkan pemahaman yang mendalam tentang bagaimana keputusan harus dibuat oleh seorang ahli manusia.

Salah satu contoh di mana pendekatan kode tangan ini akan gagal adalah dalam mendeteksi wajah dalam gambar. Saat ini, setiap smartphone dapat mendeteksi wajah dalam sebuah gambar. Namun, pendekesan wajah merupakan masalah yang belum terpecahkan hingga baru-baru ini pada tahun 2001. Masalah utamanya adalah cara piksel (yang membentuk gambar di komputer) "dipersepsikan" oleh komputer sangat berbeda dari cara manusia memandang wajah. Perbedaan representasi ini membuat pada dasarnya tidak mungkin bagi manusia untuk membuat seperangkat aturan yang baik untuk menggambarkan apa yang membentuk wajah dalam citra digital. Menggunakan pembelajaran mesin, bagaimanapun, hanya menyajikan program dengan banyak koleksi gambar wajah sudah cukup untuk algoritma untuk menentukan karakteristik apa yang diperlukan untuk mengidentifikasi wajah.

## **1.2 MASALAH YANG DAPAT DIPECAHKAN OLEH PEMBELAJARAN MESIN**

Jenis algoritma pembelajaran mesin yang paling sukses adalah yang mengotomatiskan proses pengambilan keputusan dengan menggeneralisasi dari contoh yang diketahui. Dalam pengaturan ini, yang dikenal sebagai pembelajaran terawasi, pengguna menyediakan algoritme dengan pasangan masukan dan keluaran yang diinginkan, dan algoritme menemukan cara untuk menghasilkan keluaran yang diinginkan dengan memberikan masukan. Secara khusus, algoritme mampu membuat output untuk input yang belum pernah dilihat sebelumnya tanpa bantuan manusia. Kembali ke contoh klasifikasi spam kami, menggunakan pembelajaran mesin, pengguna menyediakan algoritme dengan sejumlah besar email (yang merupakan input), bersama dengan informasi tentang apakah salah satu dari email ini adalah spam (yang merupakan output yang diinginkan). Diberikan email baru, algoritme kemudian akan menghasilkan prediksi apakah email baru tersebut adalah spam.

Algoritma pembelajaran mesin yang belajar dari pasangan input/output disebut algoritma pembelajaran terawasi karena "guru" memberikan pengawasan kepada algoritma dalam bentuk output yang diinginkan untuk setiap contoh yang mereka pelajari. Meskipun membuat kumpulan data input dan output sering kali merupakan proses manual yang melelahkan, algoritme pembelajaran yang diawasi dipahami dengan baik dan kinerjanya mudah diukur. Jika aplikasi Anda dapat dirumuskan sebagai masalah pembelajaran yang diawasi, dan Anda dapat membuat kumpulan data yang menyertakan hasil yang diinginkan, pembelajaran mesin kemungkinan akan dapat memecahkan masalah Anda.

Contoh tugas pembelajaran mesin yang diawasi meliputi:

*Mengidentifikasi kode pos dari angka tulisan tangan pada amplop*

Di sini inputnya adalah pindaian tulisan tangan, dan output yang diinginkan adalah angka sebenarnya dalam kode pos. Untuk membuat set data untuk membangun model pembelajaran mesin, Anda perlu mengumpulkan banyak amplop. Kemudian Anda dapat membaca sendiri kode pos dan menyimpan angkanya sebagai hasil yang Anda inginkan.

*Menentukan apakah tumor itu jinak berdasarkan citra medis*

Di sini inputnya adalah gambar, dan outputnya adalah apakah tumornya jinak. Untuk membuat kumpulan data untuk membangun model, Anda memerlukan database

gambar medis. Anda juga memerlukan pendapat ahli, jadi dokter perlu melihat semua gambar dan memutuskan tumor mana yang jinak dan mana yang tidak. Bahkan mungkin perlu dilakukan diagnosis tambahan di luar konten gambar untuk menentukan apakah tumor pada gambar tersebut bersifat kanker atau tidak.

#### *Mendeteksi aktivitas penipuan dalam transaksi kartu kredit*

Di sini inputnya adalah catatan transaksi kartu kredit, dan outputnya adalah kemungkinan penipuan atau tidak. Dengan asumsi bahwa Anda adalah entitas yang mendistribusikan kartu kredit, mengumpulkan dataset berarti menyimpan semua transaksi dan mencatat jika pengguna melaporkan transaksi apa pun sebagai penipuan.

Hal yang menarik untuk dicatat tentang contoh-contoh ini adalah bahwa meskipun input dan output terlihat cukup sederhana, proses pengumpulan data untuk ketiga tugas ini sangat berbeda. Meskipun membaca amplop itu melelahkan, mudah dan murah. Mendapatkan pencitraan dan diagnosis medis, di sisi lain, tidak hanya membutuhkan mesin yang mahal tetapi juga pengetahuan ahli yang langka dan mahal, belum lagi masalah etika dan masalah privasi. Dalam contoh mendeteksi penipuan kartu kredit, pengumpulan data jauh lebih sederhana. Pelanggan Anda akan memberi Anda hasil yang diinginkan, karena mereka akan melaporkan penipuan. Yang harus Anda lakukan untuk mendapatkan pasangan input/output dari aktivitas penipuan dan non-penipuan adalah menunggu.

*Algoritma tanpa pengawasan* adalah jenis algoritma lain yang akan kita bahas dalam buku ini. Dalam pembelajaran tanpa pengawasan, hanya data input yang diketahui, dan tidak ada data output yang diketahui yang diberikan ke algoritma. Meskipun ada banyak aplikasi yang berhasil dari metode ini, mereka biasanya lebih sulit untuk dipahami dan dievaluasi.

Contoh pembelajaran tanpa pengawasan meliputi:

#### *Mengidentifikasi topik dalam satu set posting blog*

Jika Anda memiliki banyak koleksi data teks, Anda mungkin ingin meringkasnya dan menemukan tema umum di dalamnya. Anda mungkin tidak tahu sebelumnya apa topik ini, atau berapa banyak topik yang mungkin ada. Oleh karena itu, tidak ada keluaran yang diketahui.

#### *Segmentasi pelanggan ke dalam kelompok dengan preferensi yang sama*

Dengan sekumpulan catatan pelanggan, Anda mungkin ingin mengidentifikasi pelanggan mana yang serupa, dan apakah ada kelompok pelanggan dengan preferensi serupa. Untuk situs belanja, ini mungkin "orang tua", "kutu buku", atau "gamer". Karena Anda tidak tahu sebelumnya apa grup ini, atau bahkan berapa jumlahnya, Anda tidak memiliki keluaran yang diketahui.

#### *Mendeteksi pola akses abnormal ke situs web*

Untuk mengidentifikasi penyalahgunaan atau bug, seringkali membantu untuk menemukan pola akses yang berbeda dari biasanya. Setiap pola abnormal mungkin sangat berbeda, dan Anda mungkin tidak memiliki contoh perilaku abnormal yang tercatat. Karena dalam contoh ini Anda hanya mengamati lalu lintas, dan Anda tidak tahu apa yang dimaksud dengan perilaku normal dan tidak normal, ini adalah masalah yang tidak diawasi.

Untuk tugas pembelajaran yang diawasi dan tidak diawasi, penting untuk memiliki representasi data input Anda yang dapat dipahami oleh komputer. Seringkali sangat membantu untuk menganggap data Anda sebagai tabel. Setiap titik data yang ingin Anda jelaskan (setiap email, setiap pelanggan, setiap transaksi) adalah satu baris, dan setiap properti yang menjelaskan titik data tersebut (misalnya, usia pelanggan atau jumlah atau lokasi transaksi) adalah kolom. Anda dapat mendeskripsikan pengguna berdasarkan usia, jenis kelamin, saat mereka membuat akun, dan seberapa sering mereka membeli dari toko online Anda. Anda dapat menggambarkan gambar tumor dengan nilai skala abu-abu setiap piksel, atau mungkin dengan menggunakan ukuran, bentuk, dan warna tumor.

Setiap entitas atau baris di sini dikenal sebagai sampel (atau titik data) dalam pembelajaran mesin, sedangkan kolom—properti yang mendeskripsikan entitas ini—disebut fitur.

Nanti dalam buku ini kita akan membahas lebih detail tentang topik membangun representasi yang baik dari data Anda, yang disebut ekstraksi fitur atau rekayasa fitur. Namun, Anda harus ingat bahwa tidak ada algoritme pembelajaran mesin yang dapat membuat prediksi pada data yang tidak memiliki informasinya. Misalnya, jika satu-satunya fitur yang Anda miliki untuk pasien adalah nama belakang mereka, tidak ada algoritme yang dapat memprediksi jenis kelamin mereka. Informasi ini sama sekali tidak terkandung dalam data Anda. Jika Anda menambahkan fitur lain yang berisi nama depan pasien, Anda akan jauh lebih beruntung, karena sering kali mungkin untuk membedakan jenis kelamin dengan nama depan seseorang.

### 1.3 MENGETAHUI TUGAS ANDA DAN MENGETAHUI DATA ANDA

Sangat mungkin bagian terpenting dalam proses pembelajaran mesin adalah memahami data yang sedang Anda kerjakan dan bagaimana kaitannya dengan tugas yang ingin Anda selesaikan. Tidak akan efektif untuk memilih algoritme secara acak dan membuang data Anda ke dalamnya. Penting untuk memahami apa yang terjadi di kumpulan data Anda sebelum Anda mulai membangun model. Setiap algoritme berbeda dalam hal jenis data dan pengaturan masalah apa yang paling cocok untuknya. Saat Anda membangun solusi pembelajaran mesin, Anda harus menjawab, atau setidaknya mengingat, pertanyaan-pertanyaan berikut:

Pertanyaan apa yang saya coba jawab? Apakah saya pikir data yang dikumpulkan dapat menjawab pertanyaan itu?

- Apa cara terbaik untuk mengungkapkan pertanyaan saya sebagai masalah pembelajaran mesin?
- Apakah saya telah mengumpulkan cukup data untuk mewakili masalah yang ingin saya pecahkan?
- Fitur data apa yang saya ekstrak, dan apakah fitur ini memungkinkan prediksi yang tepat?
- Bagaimana saya mengukur keberhasilan dalam aplikasi saya?

- Bagaimana solusi pembelajaran mesin akan berinteraksi dengan bagian lain dari penelitian atau produk bisnis saya?

Dalam konteks yang lebih besar, algoritme dan metode dalam pembelajaran mesin hanyalah satu bagian dari proses yang lebih besar untuk memecahkan masalah tertentu, dan adalah baik untuk selalu mengingat gambaran besarnya. Banyak orang menghabiskan banyak waktu untuk membangun solusi pembelajaran mesin yang kompleks, hanya untuk mengetahui bahwa mereka tidak menyelesaikan masalah yang tepat.

Ketika masuk jauh ke dalam aspek teknis pembelajaran mesin (seperti yang akan kita lakukan dalam buku ini), mudah untuk melupakan tujuan akhir. Meskipun kami tidak akan membahas pertanyaan yang tercantum di sini secara rinci, kami tetap mendorong Anda untuk mengingat semua asumsi yang mungkin Anda buat, secara eksplisit atau implisit, saat Anda mulai membuat model pembelajaran mesin.

#### **1.4 MENGAPA PYTHON?**

Python telah menjadi lingua franca untuk banyak aplikasi ilmu data. Ini menggabungkan kekuatan bahasa pemrograman tujuan umum dengan kemudahan penggunaan bahasa skrip khusus domain seperti MATLAB atau R. Python memiliki perpustakaan untuk memuat data, visualisasi, statistik, pemrosesan bahasa alami, pemrosesan gambar, dan banyak lagi. Kotak alat yang luas ini menyediakan para ilmuwan data dengan beragam fungsi tujuan umum dan khusus. Salah satu keuntungan utama menggunakan Python adalah kemampuan untuk berinteraksi langsung dengan kode, menggunakan terminal atau alat lain seperti Notebook Jupyter, yang akan kita bahas segera. Pembelajaran mesin dan analisis data adalah proses iteratif yang mendasar, di mana data mendorong analisis. Sangat penting bagi proses ini untuk memiliki alat yang memungkinkan iterasi cepat dan interaksi yang mudah.

Sebagai bahasa pemrograman tujuan umum, Python juga memungkinkan pembuatan antarmuka pengguna grafis (GUI) dan layanan web yang kompleks, dan untuk integrasi ke dalam sistem yang ada.

#### **1.5 SCIKIT-LEARN**

Scikit-learn adalah proyek sumber terbuka, artinya bebas untuk digunakan dan didistribusikan, dan siapa pun dapat dengan mudah memperoleh kode sumber untuk melihat apa yang terjadi di balik adegan. Proyek scikit-learn terus dikembangkan dan ditingkatkan, dan memiliki komunitas pengguna yang sangat aktif. Ini berisi sejumlah algoritme pembelajaran mesin canggih, serta dokumentasi komprehensif tentang setiap algoritme. scikit-learn adalah alat yang sangat populer, dan pustaka Python paling menonjol untuk pembelajaran mesin. Ini banyak digunakan di industri dan akademisi, dan banyak tutorial dan cuplikan kode tersedia secara online. Scikit-learn bekerja dengan baik dengan sejumlah alat Python ilmiah lainnya, yang akan kita bahas nanti di bab ini.

Saat membaca ini, kami menyarankan Anda juga menelusuri panduan pengguna scikit-learn dan dokumentasi API untuk detail tambahan dan lebih banyak opsi untuk setiap

algoritma. Dokumentasi online sangat menyeluruh, dan buku ini akan memberi Anda semua prasyarat dalam pembelajaran mesin untuk memahaminya secara mendetail.

### **Menginstal scikit-learn**

Scikit-learn bergantung pada dua paket Python lainnya, NumPy dan SciPy. Untuk plotting dan pengembangan interaktif, Anda juga harus menginstal matplotlib, IPython, dan Jupyter Notebook. Kami merekomendasikan untuk menggunakan salah satu dari distribusi Python yang sudah dikemas berikut, yang akan menyediakan paket-paket yang diperlukan:

#### *Anaconda*

Distribusi Python dibuat untuk pemrosesan data skala besar, analitik prediktif, dan komputasi ilmiah. Anaconda hadir dengan NumPy, SciPy, matplotlib, pandas, IPython, Jupyter Notebook, dan scikit-learn. Tersedia di Mac OS, Windows, dan Linux, ini adalah solusi yang sangat nyaman dan merupakan solusi yang kami sarankan untuk orang-orang yang tidak memiliki instalasi paket Python ilmiah. Anaconda sekarang juga menyertakan perpustakaan Intel MKL komersial secara gratis. Menggunakan MKL (yang dilakukan secara otomatis ketika Anaconda diinstal) dapat memberikan peningkatan kecepatan yang signifikan untuk banyak algoritma dalam scikit-learn.

#### *Kanopi yang Dipikirkan*

Distribusi Python lain untuk komputasi ilmiah. Ini hadir dengan NumPy, SciPy, matplotlib, pandas, dan IPython, tetapi versi gratisnya tidak disertakan dengan scikit-learn. Jika Anda adalah bagian dari institusi akademik, pemberi gelar, Anda dapat meminta lisensi akademik dan mendapatkan akses gratis ke versi berlangganan berbayar dari Enthought Canopy. Enthought Canopy tersedia untuk Python 2.7.x, dan berfungsi di Mac OS, Windows, dan Linux.

#### *Python(x,y)*

Distribusi Python gratis untuk komputasi ilmiah, khusus untuk Windows. Python(x,y) hadir dengan NumPy, SciPy, matplotlib, pandas, IPython, dan scikit-learn.

Jika Anda sudah menyiapkan instalasi Python, Anda dapat menggunakan pip untuk menginstal semua paket ini:

```
Rp pip install numpy scipy matplotlib ipython scikit-learn panda
```

## **1.6 PERPUSTAKAAN DAN ALAT PENTING**

Memahami apa itu scikit-learn dan bagaimana menggunakan menggunakannya adalah penting, tetapi ada beberapa perpustakaan lain yang akan meningkatkan pengalaman Anda. scikit-learn dibangun di atas pustaka Python ilmiah NumPy dan SciPy. Selain NumPy dan SciPy, kita akan menggunakan pandas dan matplotlib. Kami juga akan memperkenalkan Jupyter Notebook, yang merupakan lingkungan pemrograman interaktif berbasis browser. Secara singkat, inilah yang harus Anda ketahui tentang alat-alat ini untuk mendapatkan hasil maksimal dari scikit-learn.<sup>1</sup>

### **Buku Catatan Jupyter**

Notebook Jupyter adalah lingkungan interaktif untuk menjalankan kode di browser. Ini adalah alat yang hebat untuk analisis data eksplorasi dan banyak digunakan oleh para ilmuwan data. Sementara Notebook Jupyter mendukung banyak bahasa pemrograman, kami hanya

membutuhkan dukungan Python. Notebook Jupyter memudahkan untuk memasukkan kode, teks, dan gambar, dan semua buku ini sebenarnya ditulis sebagai Notebook Jupyter. Semua contoh kode yang kami sertakan dapat diunduh dari GitHub.

### JumlahPy

NumPy adalah salah satu paket dasar untuk komputasi ilmiah dengan Python. Ini berisi fungsionalitas untuk array multidimensi, fungsi matematika tingkat tinggi seperti operasi aljabar linier dan transformasi Fourier, dan generator nomor pseudorandom.

Dalam scikit-learn, array NumPy adalah struktur data fundamental. scikit-learn mengambil data dalam bentuk array NumPy. Data apa pun yang Anda gunakan harus dikonversi ke array NumPy. Fungsionalitas inti NumPy adalah kelas ndarray, array multidimensi (n-dimensi). Semua elemen array harus bertipe sama. Array NumPy terlihat seperti ini:

**In[2]:**

```
import numpy as np

x = np.array([[1, 2, 3], [4, 5, 6]])
print("x:\n{}".format(x))
```

**Out[2]:**

```
x:
[[1 2 3]
 [4 5 6]]
```

Kami akan banyak menggunakan NumPy dalam buku ini, dan kami akan merujuk ke objek kelas ndarray NumPy sebagai "array NumPy" atau hanya "array."

### SciPy

SciPy adalah kumpulan fungsi untuk komputasi ilmiah dengan Python. Ini menyediakan, antara lain fungsionalitas, rutinitas aljabar linier canggih, optimasi fungsi matematika, pemrosesan sinyal, fungsi matematika khusus, dan distribusi statistik. scikit-learn mengambil dari kumpulan fungsi SciPy untuk mengimplementasikan algoritmenya. Bagian terpenting dari SciPy bagi kami adalah `scipy.sparse`: ini menyediakan matriks sparse, yang merupakan representasi lain yang digunakan untuk data dalam scikit-belajar. Matriks jarang digunakan setiap kali kita ingin menyimpan larik 2D yang sebagian besar berisi nol:

**In[3]:**

```
from scipy import sparse

# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("NumPy array:\n{}".format(eye))
```

**Out[3]:**

```
NumPy array:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

**In[4]:**

```
# Convert the NumPy array to a SciPy sparse matrix in CSR format
# Only the nonzero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nSciPy sparse CSR matrix:\n{}".format(sparse_matrix))
```

**Out[4]:**

```
SciPy sparse CSR matrix:
 (0, 0)    1.0
 (1, 1)    1.0
 (2, 2)    1.0
 (3, 3)    1.0
```

Biasanya tidak mungkin untuk membuat representasi padat dari data yang jarang (karena tidak sesuai dengan memori), jadi kita perlu membuat representasi yang jarang secara langsung. Berikut adalah cara untuk membuat matriks sparse yang sama seperti sebelumnya, menggunakan format COO:

**In[5]:**

```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("COO representation:\n{}".format(eye_coo))
```

**Out[5]:**

```
COO representation:
 (0, 0)    1.0
 (1, 1)    1.0
 (2, 2)    1.0
 (3, 3)    1.0
```

Rincian lebih lanjut tentang matriks jarang SciPy dapat ditemukan di Catatan Kuliah SciPy.

## matplotlib

matplotlib adalah pustaka plot ilmiah utama dengan Python. Ini menyediakan fungsi untuk membuat visualisasi kualitas publikasi seperti diagram garis, histogram, plot sebar, dan sebagainya. Memvisualisasikan data Anda dan berbagai aspek analisis Anda dapat memberi Anda wawasan penting, dan kami akan menggunakan matplotlib untuk semua visualisasi

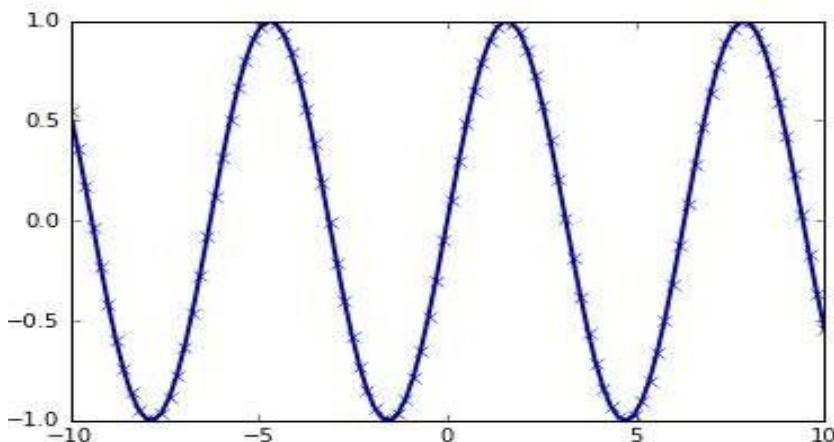
*Machine Learning (Dr. Budi Raharjo)*

kami. Saat bekerja di dalam Notebook Jupyter, Anda dapat menampilkan angka secara langsung di browser dengan menggunakan %matplotlib notebook dan %matplotlib perintah inline. Kami merekomendasikan penggunaan %matplotlib notebook, yang menyediakan lingkungan interaktif (meskipun kami menggunakan %matplotlib inline untuk menghasilkan buku ini). Misalnya, kode ini menghasilkan plot pada Gambar 1.1:

In[6]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```



**Gambar 1.1** Plot garis sederhana dari fungsi sinus menggunakan matplotlib

## Panda

pandas adalah pustaka Python untuk perselisihan dan analisis data. Itu dibangun di sekitar struktur data yang disebut DataFrame yang dimodelkan setelah R DataFrame. Sederhananya, pandas DataFrame adalah tabel, mirip dengan spreadsheet Excel. pandas menyediakan berbagai macam metode untuk memodifikasi dan mengoperasikan tabel ini; khususnya, ini memungkinkan kueri dan gabungan tabel seperti SQL. Berbeda dengan NumPy, yang mengharuskan semua entri dalam array memiliki tipe yang sama, panda memungkinkan setiap kolom memiliki tipe terpisah (misalnya, bilangan bulat, tanggal, angka titik-mengambang, dan string). Alat berharga lain yang disediakan oleh panda adalah kemampuannya untuk menyerap dari berbagai macam format file dan basis data, seperti file SQL, Excel, dan file comma-separated values (CSV). Mendetail tentang fungsi panda tidak termasuk dalam cakupan buku ini. Namun, Python untuk Analisis Data oleh Wes McKinney (O'Reilly, 2012) memberikan panduan yang bagus. Berikut adalah contoh kecil membuat DataFrame menggunakan kamus:

In[7]:

```
import pandas as pd

# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
        'Location' : ["New York", "Paris", "Berlin", "London"],
        'Age' : [24, 13, 53, 33]
       }

data_pandas = pd.DataFrame(data)
# IPython.display allows "pretty printing" of dataframes
# in the Jupyter notebook
display(data_pandas)
```

Ini menghasilkan output berikut:

	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

Ada beberapa cara yang mungkin untuk menanyakan tabel ini. Sebagai contoh:

In[8]:

```
# Select all rows that have an age column greater than 30
display(data_pandas[data_pandas.Age > 30])
```

Ini menghasilkan hasil berikut:

	Age	Location	Name
2	53	Berlin	Peter
3	33	London	Linda

## mglearn

Buku ini dilengkapi dengan kode yang menyertainya, yang dapat Anda temukan di GitHub. Kode terlampir tidak hanya mencakup semua contoh yang ditampilkan dalam buku ini, tetapi juga perpustakaan mglearn. Ini adalah perpustakaan fungsi utilitas yang kami tulis untuk buku ini, sehingga kami tidak mengacaukan daftar kode kami dengan detail plot dan pemutuan data. Jika Anda tertarik, Anda dapat mencari semua fungsi di repositori, tetapi detail modul mglearn tidak terlalu penting untuk materi dalam buku ini. Jika Anda melihat panggilan ke mglearn dalam kode, biasanya itu adalah cara untuk membuat gambar yang bagus dengan cepat, atau untuk mendapatkan beberapa data menarik.



Sepanjang buku ini kami banyak menggunakan NumPy, matplotlib dan pandas. Semua kode akan mengasumsikan impor berikut:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
```

Kami juga berasumsi bahwa Anda akan menjalankan kode di Jupyter Notebook dengan %matplotlib notebook atau %matplotlib inline magic diaktifkan untuk menampilkan plot. Jika Anda tidak menggunakan buku catatan atau perintah ajaib ini, Anda harus memanggil plt.show untuk benar-benar menampilkan salah satu angka.

### Python 2 Versus Python 3

Ada dua versi utama Python yang banyak digunakan saat ini: Python 2 (lebih tepatnya, 2.7) dan Python 3 (dengan rilis terbaru 3.5 pada saat penulisan). Hal ini terkadang menyebabkan beberapa kebingungan. Python 2 tidak lagi aktif dikembangkan, tetapi karena Python 3 berisi perubahan besar, kode Python 2 biasanya tidak berjalan di Python 3. Jika Anda baru mengenal Python, atau memulai proyek baru dari awal, kami sangat menyarankan menggunakan versi terbaru dari Python 3 tanpa perubahan. Jika Anda memiliki basis kode besar yang Anda andalkan yang ditulis untuk Python 2, Anda tidak diperbolehkan memutakhirkan untuk saat ini. Namun, Anda harus mencoba bermigrasi ke Python 3 sesegera mungkin. Saat menulis kode baru, sebagian besar cukup mudah untuk menulis kode yang berjalan di bawah Python 2 dan Python 3.2. Jika Anda tidak harus berinteraksi dengan perangkat lunak lama, Anda harus menggunakan Python 3. Semua kode dalam buku ini ditulis dengan cara yang bekerja untuk kedua versi. Namun, output yang tepat mungkin sedikit berbeda di bawah Python 2.

## 1.7 VERSI YANG DIGUNAKAN DALAM BUKU INI

Kami menggunakan versi berikut dari perpustakaan yang disebutkan sebelumnya dalam buku ini:

In[9]:

```
import sys
print("Python version: {}".format(sys.version))

import pandas as pd
print("pandas version: {}".format(pd.__version__))

import matplotlib
print("matplotlib version: {}".format(matplotlib.__version__))

import numpy as np
print("NumPy version: {}".format(np.__version__))

import scipy as sp
print("SciPy version: {}".format(sp.__version__))

import IPython
print("IPython version: {}".format(IPython.__version__))

import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
```

**Out[9]:**

```
Python version: 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul 2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
pandas version: 0.18.1
matplotlib version: 1.5.1
NumPy version: 1.11.1
SciPy version: 0.17.1
IPython version: 5.1.0
scikit-learn version: 0.18
```

Meskipun tidak penting untuk mencocokkan versi ini dengan tepat, Anda harus memiliki versi scikit-learn yang paling baru seperti yang kami gunakan. Sekarang setelah kita menyiapkan semuanya, mari selami aplikasi pembelajaran mesin pertama kita.

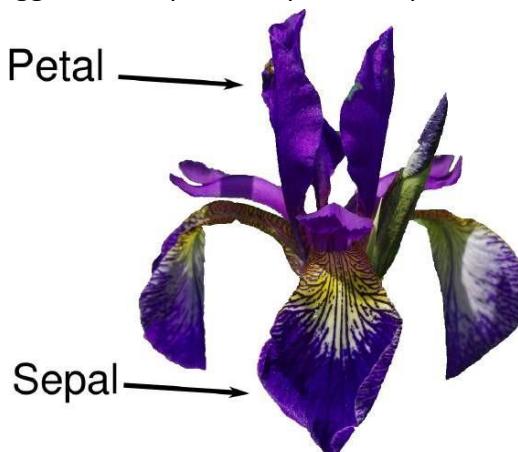


Buku ini mengasumsikan bahwa Anda memiliki scikit-belajar versi 0.18 atau lebih baru. Modul `model_selection` ditambahkan di 0.18, dan jika Anda menggunakan versi scikit-learn yang lebih lama, Anda perlu menyesuaikan impor dari modul ini.

## 1.8 APLIKASI PERTAMA: MENGKLASIFIKASIKAN SPESIES IRIS

Di bagian ini, kita akan melalui aplikasi pembelajaran mesin sederhana dan membuat model pertama kita. Dalam prosesnya, kami akan memperkenalkan beberapa konsep dan istilah inti. Mari kita asumsikan bahwa seorang ahli botani yang hobi tertarik untuk membedakan spesies beberapa bunga iris yang dia temukan. Dia telah mengumpulkan beberapa pengukuran yang terkait dengan setiap iris: panjang dan lebar kelopak bunga dan panjang dan lebar kelopak bunga, semuanya diukur dalam sentimeter (lihat Gambar 1.2).

Dia juga memiliki ukuran beberapa iris yang sebelumnya telah diidentifikasi oleh ahli botani sebagai milik spesies setosa, versicolor, atau virginica. Untuk pengukuran ini, dia dapat memastikan spesies mana yang dimiliki setiap iris. Mari kita asumsikan bahwa ini adalah satu-satunya spesies yang akan ditemui ahli botani hobi kita di alam liar. Tujuan kami adalah membangun model pembelajaran mesin yang dapat belajar dari pengukuran iris yang spesiesnya diketahui, sehingga kami dapat memprediksi spesies untuk iris baru.



Gambar 1.2 Bagian dari bunga iris

Karena kami memiliki pengukuran yang kami ketahui spesies iris yang benar, ini adalah masalah pembelajaran yang diawasi. Dalam masalah ini, kami ingin memprediksi salah satu dari beberapa opsi (spesies iris). Ini adalah contoh dari masalah klasifikasi. Kemungkinan keluaran (spesies iris yang berbeda) disebut kelas. Setiap iris dalam dataset milik salah satu dari tiga kelas, jadi masalah ini adalah masalah klasifikasi tiga kelas.

Output yang diinginkan untuk satu titik data (iris) adalah spesies bunga ini. Untuk titik data tertentu, spesies yang dimilikinya disebut labelnya.

### Kenali Datanya

Data yang akan kita gunakan untuk contoh ini adalah dataset Iris, dataset klasik dalam pembelajaran mesin dan statistik. Itu termasuk dalam scikit-learn dalam modul datasets. Kita dapat memuatnya dengan memanggil fungsi `load_iris`:

**In[10]:**

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Objek iris yang dikembalikan oleh `load_iris` adalah objek `Bunch`, yang sangat mirip dengan kamus. Ini berisi kunci dan nilai:

**In[11]:**

```
print("Keys of iris_dataset: \n{}".format(iris_dataset.keys()))
```

**Out[11]:**

```
Keys of iris_dataset:
dict_keys(['target_names', 'feature_names', 'DESCR', 'data', 'target'])
```

Nilai kunci `DESCR` adalah deskripsi singkat dari kumpulan data. Kami menunjukkan awal deskripsi di sini (jangan ragu untuk mencari sendiri sisanya):

**In[12]:**

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

**Out[12]:**

```
Iris Plants Database
=====
Notes
-----
Data Set Characteristics:
 :Number of Instances: 150 (50 in each of three classes)
 :Number of Attributes: 4 numeric, predictive att
 ...
-----
```

Nilai dari key `target_names` adalah array string, yang berisi spesies bunga yang ingin kita prediksi:

**In[13]:**

```
print("Target names: {}".format(iris_dataset['target_names']))
```

**Out[13]:**

```
Target names: ['setosa' 'versicolor' 'virginica']
```

Nilai feature\_names adalah daftar string, memberikan deskripsi setiap fitur:

**In[14]:**

```
print("Feature names: \n{}".format(iris_dataset['feature_names']))
```

**Out[14]:**

```
Feature names:  
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',  
'petal width (cm)']
```

Data itu sendiri terkandung di bidang target dan data. data berisi pengukuran numerik panjang sepal, lebar sepal, panjang petal, dan lebar petal dalam array NumPy:

**In[15]:**

```
print("Type of data: {}".format(type(iris_dataset['data'])))
```

**Out[15]:**

```
Type of data: <class 'numpy.ndarray'>
```

Baris dalam larik data sesuai dengan bunga, sedangkan kolom mewakili empat pengukuran yang dilakukan untuk setiap bunga:

**In[16]:**

```
print("Shape of data: {}".format(iris_dataset['data'].shape))
```

**Out[16]:**

```
Shape of data: (150, 4)
```

Kita melihat bahwa susunan itu berisi ukuran untuk 150 bunga yang berbeda. Ingatlah bahwa masing-masing item disebut sampel dalam pembelajaran mesin, dan propertinya disebut fitur. Bentuk array data adalah jumlah sampel dikalikan dengan jumlah fitur. Ini adalah konvensi dalam scikit-learn, dan data Anda akan selalu dianggap dalam bentuk ini. Berikut adalah nilai fitur untuk lima sampel pertama:

**In[17]:**

```
print("First five columns of data:\n{}".format(iris_dataset['data'][:5]))
```

**Out[17]:**

```
First five columns of data:  
[[ 5.1  3.5  1.4  0.2]  
 [ 4.9  3.   1.4  0.2]  
 [ 4.7  3.2  1.3  0.2]  
 [ 4.6  3.1  1.5  0.2]  
 [ 5.   3.6  1.4  0.2]]
```

Dari data ini kita dapat melihat bahwa kelima bunga pertama memiliki lebar kelopak 0,2 cm dan bunga pertama memiliki sepal terpanjang, yaitu 5,1 cm.

Larik target berisi spesies masing-masing bunga yang diukur, juga sebagai larik NumPy:

**In[18]:**

```
print("Type of target: {}".format(type(iris_dataset['target'])))
```

**Out[18]:**

```
Type of target: <class 'numpy.ndarray'>
```

target adalah larik satu dimensi, dengan satu entri per bunga:

**In[19]:**

```
print("Shape of target: {}".format(iris_dataset['target'].shape))
```

**Out[19]:**

```
Shape of target: (150,)
```

Spesies dikodekan sebagai bilangan bulat dari 0 hingga 2:

**In[20]:**

```
print("Target:\n{}".format(iris_dataset['target']))
```

**Out[20]:**

```
Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Arti angka diberikan oleh array `iris['target_names']`: 0 berarti setosa, 1 berarti versicolor, dan 2 berarti virginica.

## 1.9 MENGUKUR KEBERHASILAN: DATA PELATIHAN DAN PENGUJIAN

Kami ingin membangun model pembelajaran mesin dari data ini yang dapat memprediksi spesies iris untuk serangkaian pengukuran baru. Tetapi sebelum kita dapat menerapkan model kita ke pengukuran baru, kita perlu tahu apakah model itu benar-benar berfungsi—yaitu, apakah kita harus memercayai prediksinya.

Sayangnya, kami tidak dapat menggunakan data yang kami gunakan untuk membangun model untuk mengevaluasinya. Ini karena model kami selalu dapat dengan mudah mengingat seluruh set pelatihan, dan karena itu akan selalu memprediksi label yang benar untuk setiap titik dalam set pelatihan. "Pengingat" ini tidak menunjukkan kepada kita apakah model kita akan digeneralisasi dengan baik (dengan kata lain, apakah model itu juga akan berkinerja baik pada data baru).

Untuk menilai kinerja model, kami menunjukkan data baru (data yang belum pernah dilihat sebelumnya) yang kami beri label. Ini biasanya dilakukan dengan membagi data berlabel yang telah kami kumpulkan (di sini, 150 pengukuran bunga kami) menjadi dua bagian. Satu bagian dari data digunakan untuk membangun model pembelajaran mesin kami, dan disebut data pelatihan atau set pelatihan. Sisa data akan digunakan untuk menilai seberapa baik model bekerja; ini disebut data uji, set uji, atau set tahan.

Scikit-learn berisi fungsi yang mengacak dataset dan membaginya untuk Anda: fungsi `train_test_split`. Fungsi ini mengekstrak 75% baris dalam data sebagai set pelatihan, bersama dengan label yang sesuai untuk data ini. Sisa 25% dari data, bersama dengan label yang tersisa, dinyatakan sebagai set pengujian. Memutuskan berapa banyak data yang ingin Anda masukkan ke dalam pelatihan dan set tes masing-masing agak sewenang-wenang, tetapi menggunakan set tes yang berisi 25% dari data adalah aturan praktis yang baik.

Dalam scikit-learn, data biasanya dilambangkan dengan huruf besar X, sedangkan label dilambangkan dengan huruf kecil y. Ini terinspirasi oleh rumusan standar  $f(x)=y$  dalam matematika, di mana x adalah input ke suatu fungsi dan y adalah outputnya. Mengikuti lebih banyak konvensi dari matematika, kami menggunakan huruf besar X karena datanya adalah array dua dimensi (matriks) dan huruf kecil y karena targetnya adalah array satu dimensi (vektor).

Mari kita panggil `train_test_split` pada data kita dan tetapkan outputnya menggunakan nomenklatur ini:

**In[21]:**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

Sebelum membuat pemisahan, fungsi `train_test_split` mengacak dataset menggunakan generator angka pseudorandom. Jika kita hanya mengambil 25% terakhir dari data sebagai set pengujian, semua titik data akan memiliki label 2, karena titik data diurutkan berdasarkan label (lihat output untuk `iris['target']` yang ditunjukkan sebelumnya). Menggunakan set tes yang hanya berisi satu dari tiga kelas tidak akan memberi tahu kami banyak tentang seberapa baik model kami digeneralisasi, jadi kami mengacak data kami untuk memastikan data uji berisi data dari semua kelas.

Untuk memastikan bahwa kita akan mendapatkan output yang sama jika kita menjalankan fungsi yang sama beberapa kali, kita menyediakan generator nomor pseudorandom dengan seed tetap menggunakan parameter `random_state`. Ini akan membuat hasilnya deterministik, sehingga garis ini akan selalu memiliki hasil yang sama. Kami akan selalu memperbaiki `random_state` dengan cara ini saat menggunakan prosedur acak dalam buku ini.

Output dari fungsi `train_test_split` adalah `X_train`, `X_test`, `y_train`, dan `y_test`, yang semuanya adalah array NumPy. `X_train` berisi 75% dari baris dataset, dan `X_test` berisi 25% sisanya:

**In[22]:**

```
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
```

**Out[22]:**

```
X_train shape: (112, 4)
y_train shape: (112,)
```

## 1.10 HAL PERTAMA YANG PERTAMA: LIHAT DATA ANDA

Sebelum membangun model pembelajaran mesin, sering kali merupakan ide yang baik untuk memeriksa data, untuk melihat apakah tugas dapat diselesaikan dengan mudah tanpa pembelajaran mesin, atau jika informasi yang diinginkan mungkin tidak terkandung dalam data. Selain itu, memeriksa data Anda adalah cara yang baik untuk menemukan kelainan dan keanehan. Mungkin beberapa iris Anda diukur menggunakan inci dan bukan sentimeter, misalnya. Di dunia nyata, inkonsistensi dalam data dan pengukuran yang tidak terduga sangat umum terjadi.

Salah satu cara terbaik untuk memeriksa data adalah dengan memvisualisasikannya. Salah satu cara untuk melakukan ini adalah dengan menggunakan plot pencar. Plot sebar data menempatkan satu fitur di sepanjang sumbu x dan lainnya di sepanjang sumbu y, dan menggambar sebuah titik untuk setiap titik data. Sayangnya, layar komputer hanya memiliki dua dimensi, yang memungkinkan kita untuk memplot hanya dua (atau mungkin tiga) fitur sekaligus. Sulit untuk memplot kumpulan data dengan lebih dari tiga fitur dengan cara ini. Salah satu cara mengatasi masalah ini adalah dengan melakukan plot pasangan, yang melihat semua kemungkinan pasangan fitur. Jika Anda memiliki sejumlah kecil fitur, seperti empat yang kami miliki di sini, ini cukup masuk akal. Namun, Anda harus ingat bahwa plot berpasangan tidak menunjukkan interaksi semua fitur sekaligus, sehingga beberapa aspek menarik dari data mungkin tidak terungkap saat memvisualisasikannya dengan cara ini.

Gambar 1.3 adalah plot pasangan fitur dalam set pelatihan. Titik data diwarnai sesuai dengan spesies yang dimiliki iris. Untuk membuat plot, pertama-tama kita ubah array NumPy menjadi pandas DataFrame. pandas memiliki fungsi untuk membuat plot pasangan yang disebut scatter\_matrix. Diagonal matriks ini diisi dengan histogram dari setiap fitur:

**In[23]:**

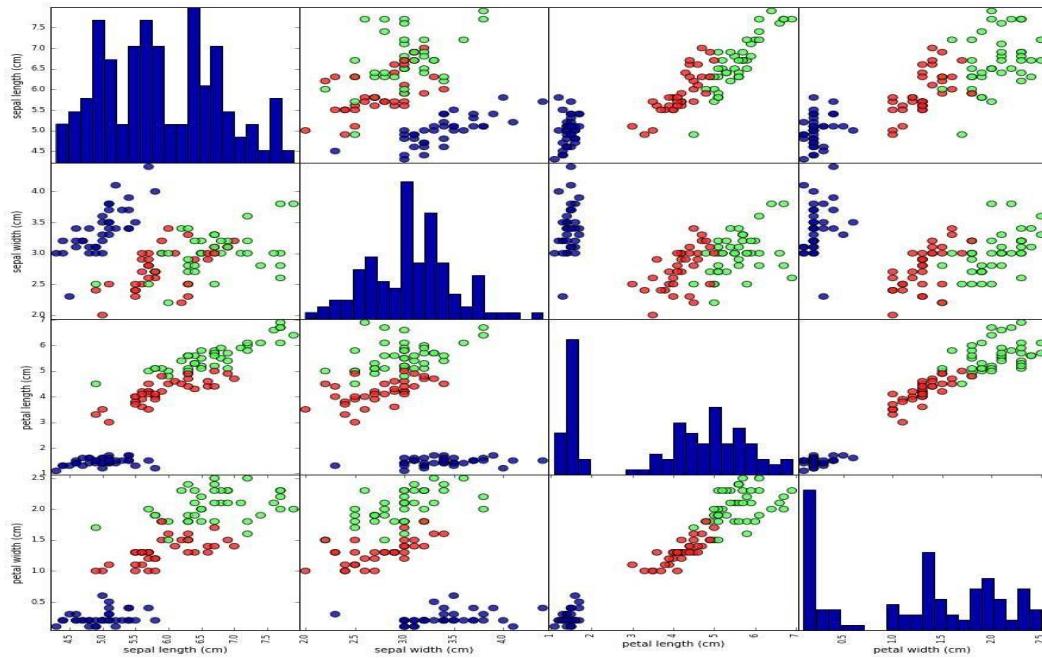
```
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))
```

**Out[23]:**

```
X_test shape: (38, 4)
y_test shape: (38,)
```

**In[24]:**

```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
grr = pd.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                        hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```



**Gambar 1.3** Plot pasangan dari dataset Iris, diwarnai oleh label kelas

Dari plot, kita dapat melihat bahwa ketiga kelas tampak terpisah dengan baik menggunakan ukuran sepal dan petal. Ini berarti bahwa model pembelajaran mesin kemungkinan akan dapat belajar memisahkannya.

### 1.11 MEMBANGUN MODEL PERTAMA ANDA: *K-NEAREST NEIGHBORS*

Sekarang kita dapat mulai membangun model pembelajaran mesin yang sebenarnya. Ada banyak algoritma klasifikasi dalam scikit-learn yang bisa kita gunakan. Di sini kita akan menggunakan classifier k-nearest neighbor, yang mudah dipahami. Membangun model ini hanya terdiri dari menyimpan set pelatihan. Untuk membuat prediksi untuk titik data baru, algoritma menemukan titik dalam set pelatihan yang paling dekat dengan titik baru. Kemudian memberikan label titik pelatihan ini ke titik data baru.

K di k-nearest tetangga menandakan bahwa alih-alih hanya menggunakan tetangga terdekat ke titik data baru, kita dapat mempertimbangkan sejumlah k tetangga dalam pelatihan (misalnya, tiga atau lima tetangga terdekat). Kemudian, kita dapat membuat prediksi menggunakan kelas mayoritas di antara tetangga-tetangga ini. Kami akan membahas lebih detail tentang ini di Bab 2; untuk saat ini, kami hanya akan menggunakan satu tetangga.

Semua model pembelajaran mesin di scikit-learn diimplementasikan di kelasnya masing-masing, yang disebut kelas Estimator. Algoritma klasifikasi k-nearest neighbor diimplementasikan di kelas KNeighborsClassifier di modul tetangga. Sebelum kita dapat menggunakan model, kita perlu membuat instance kelas menjadi objek. Ini adalah saat kita akan mengatur parameter model apa pun. Parameter terpenting KNeighborsClassifier adalah jumlah tetangga, yang akan kita atur ke 1:

**In[25]:**

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Objek knn merangkum algoritma yang akan digunakan untuk membangun model dari data pelatihan, serta algoritma untuk membuat prediksi pada titik data baru. Ini juga akan menyimpan informasi yang telah diekstraksi oleh algoritma dari data pelatihan. Dalam kasus KNeighborsClassifier, itu hanya akan menyimpan set pelatihan.

Untuk membangun model pada set pelatihan, kita memanggil metode fit dari objek knn, yang mengambil argumen array NumPy X\_train yang berisi data pelatihan dan array NumPy y\_train dari label pelatihan yang sesuai:

**In[26]:**

```
knn.fit(X_train, y_train)
```

**Out[26]:**

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                     weights='uniform')
```

Metode fit mengembalikan objek knn itu sendiri (dan memodifikasinya di tempatnya), jadi kita mendapatkan representasi string dari classifier kita. Representasi menunjukkan kepada kita parameter mana yang digunakan dalam membuat model. Hampir semuanya adalah nilai default, tetapi Anda juga dapat menemukan n\_neighbors=1, yang merupakan parameter yang kami lewati.

Sebagian besar model dalam scikit-learn memiliki banyak parameter, tetapi sebagian besar adalah pengoptimalan kecepatan atau untuk kasus penggunaan yang sangat khusus. Anda tidak perlu khawatir tentang parameter lain yang ditampilkan dalam representasi ini. Mencetak model scikit-learn dapat menghasilkan string yang sangat panjang, tetapi jangan terintimidasi oleh ini. Kami akan membahas semua parameter penting di Bab 2. Di sisa buku ini, kami tidak akan menampilkan output fit karena tidak mengandung informasi baru.

## 1.12 MEMBUAT PREDIKSI

Sekarang kita dapat membuat prediksi menggunakan model ini pada data baru yang mungkin tidak kita ketahui labelnya dengan benar. Bayangkan kita menemukan iris di alam liar dengan panjang sepals 5 cm, lebar sepals 2,9 cm, panjang kelopak 1 cm, dan lebar kelopak 0,2 cm. Jenis iris apakah ini? Kita dapat memasukkan data ini ke dalam array NumPy, sekali lagi dengan menghitung bentuknya—yaitu, jumlah sampel (1) dikalikan dengan jumlah fitur (4):

**In[27]:**

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

**Out[27]:**

```
X_new.shape: (1, 4)
```

Perhatikan bahwa kami membuat pengukuran bunga tunggal ini menjadi satu baris dalam array NumPy dua dimensi, karena scikit-learn selalu mengharapkan array dua dimensi untuk datanya. Untuk membuat prediksi, kami memanggil metode prediksi dari objek knn:

**In[28]:**

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(
    iris_dataset['target_names'][prediction]))
```

**Out[28]:**

```
Prediction: [0]
Predicted target name: ['setosa']
```

Model kami memprediksi bahwa iris baru ini termasuk dalam kelas 0, artinya spesiesnya adalah setosa. Tapi bagaimana kita tahu apakah kita bisa mempercayai model kita? Kita tidak tahu spesies yang tepat dari sampel ini, yang merupakan inti dari pembuatan model!

### 1.13 MENGEVALUASI MODEL

Di sinilah test set yang kita buat sebelumnya masuk. Data ini tidak digunakan untuk membangun model, tapi kita tahu spesies yang benar untuk setiap iris di test set. Oleh karena itu, kami dapat membuat prediksi untuk setiap iris dalam data uji dan membandingkannya dengan labelnya (spesies yang diketahui). Kita dapat mengukur seberapa baik model bekerja dengan menghitung akurasi, yang merupakan fraksi bunga yang diprediksi spesiesnya tepat:

**In[29]:**

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
```

**Out[29]:**

```
Test set predictions:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
```

**In[30]:**

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

**Out[30]:**

```
Test set score: 0.97
```

Kami juga dapat menggunakan metode skor objek knn, yang akan menghitung akurasi set tes untuk kami:

**In[31]:**

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

**Out[31]:**

```
Test set score: 0.97
```

Untuk model ini, akurasi set pengujian adalah sekitar 0,97, yang berarti kami membuat prediksi yang tepat untuk 97% iris dalam set pengujian. Di bawah beberapa asumsi matematis,

ini berarti bahwa kita dapat mengharapkan model kita benar 97% dari waktu untuk iris baru. Untuk aplikasi ahli botani hobi kami, tingkat akurasi yang tinggi ini berarti bahwa model kami mungkin cukup dapat dipercaya untuk digunakan. Dalam bab selanjutnya kita akan membahas bagaimana kita dapat meningkatkan kinerja, dan peringatan apa yang ada dalam menyetel model.

#### **1.14 RINGKASAN DAN PANDANGAN**

Mari kita rangkum apa yang kita pelajari dalam bab ini. Kami mulai dengan pengenalan singkat tentang pembelajaran mesin dan aplikasinya, kemudian membahas perbedaan antara pembelajaran terawasi dan tidak terawasi dan memberikan gambaran umum tentang alat yang akan kami gunakan dalam buku ini. Kemudian, kami merumuskan tugas memprediksi spesies iris mana yang dimiliki bunga tertentu dengan menggunakan pengukuran fisik bunga. Kami menggunakan kumpulan data pengukuran yang dianotasi oleh seorang ahli dengan spesies yang benar untuk membangun model kami, menjadikannya tugas pembelajaran yang diawasi. Ada tiga kemungkinan spesies, setosa, versicolor, atau virginica, yang membuat tugas ini menjadi masalah klasifikasi tiga kelas. Spesies yang mungkin disebut kelas dalam masalah klasifikasi, dan spesies dari iris tunggal disebut labelnya.

Dataset Iris terdiri dari dua array NumPy: satu berisi data, yang disebut sebagai  $X$  dalam scikit-learn, dan satu berisi output yang benar atau diinginkan, yang disebut  $y$ . Larik  $X$  adalah larik fitur dua dimensi, dengan satu baris per titik data dan satu kolom per fitur. Array  $y$  adalah array satu dimensi, yang di sini berisi satu label kelas, bilangan bulat mulai dari 0 hingga 2, untuk masing-masing sampel.

Kami membagi dataset kami menjadi satu set pelatihan, untuk membangun model kami, dan satu set pengujian, untuk mengevaluasi seberapa baik model kami akan digeneralisasi ke data baru yang sebelumnya tidak terlihat.

Kami memilih algoritma klasifikasi k-nearest neighbor, yang membuat prediksi untuk titik data baru dengan mempertimbangkan tetangga terdekatnya dalam set pelatihan. Ini diimplementasikan di kelas KNeighborsClassifier, yang berisi algoritme yang membangun model serta algoritme yang membuat prediksi menggunakan model. Kami membuat instance kelas, mengatur parameter. Kemudian kami membangun model dengan memanggil metode fit, meneruskan data pelatihan ( $X\_train$ ) dan output pelatihan ( $y\_train$ ) sebagai parameter. Kami mengevaluasi model menggunakan metode skor, yang menghitung akurasi model. Kami menerapkan metode skor pada data set pengujian dan label set pengujian dan menemukan bahwa model kami sekitar 97% akurat, artinya benar 97% dari waktu pada set pengujian.

Ini memberi kami kepercayaan diri untuk menerapkan model ke data baru (dalam contoh kami, pengukuran bunga baru) dan percaya bahwa model akan benar sekitar 97% dari waktu.

Berikut adalah ringkasan kode yang diperlukan untuk keseluruhan prosedur pelatihan dan evaluasi:

**In[32]:**

```
X_train, X_test, y_train, y_test = train_test_split(  
    iris_dataset['data'], iris_dataset['target'], random_state=0)  
  
knn = KNeighborsClassifier(n_neighbors=1)  
knn.fit(X_train, y_train)  
  
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

**Out[32]:**

```
Test set score: 0.97
```

Cuplikan ini berisi kode inti untuk menerapkan algoritme pembelajaran mesin apa pun menggunakan scikit-learn. Metode kecocokan, prediksi, dan skor adalah antarmuka umum untuk model yang diawasi dalam scikit-learn, dan dengan konsep yang diperkenalkan dalam bab ini, Anda dapat menerapkan model ini ke banyak tugas pembelajaran mesin. Pada bab berikutnya, kita akan membahas lebih dalam tentang berbagai jenis model terawasi dalam scikit-learn dan bagaimana menerapkannya dengan sukses.

## BAB 2

### PEMBELAJARAN TERAWASI

Seperti yang kami sebutkan sebelumnya, pembelajaran mesin yang diawasi adalah salah satu jenis pembelajaran mesin yang paling umum digunakan dan berhasil. Dalam bab ini, kami akan menjelaskan pembelajaran terawasi secara lebih rinci dan menjelaskan beberapa algoritma pembelajaran terawasi yang populer. Kita sudah melihat aplikasi pembelajaran mesin yang diawasi di Bab 1: mengklasifikasikan bunga iris menjadi beberapa spesies menggunakan pengukuran fisik bunga.

Ingin bahwa pembelajaran terawasi digunakan setiap kali kita ingin memprediksi hasil tertentu dari input yang diberikan, dan kita memiliki contoh pasangan input/output. Kami membangun model pembelajaran mesin dari pasangan input/output ini, yang terdiri dari set pelatihan kami. Tujuan kami adalah membuat prediksi akurat untuk data baru yang belum pernah dilihat sebelumnya. Pembelajaran yang diawasi seringkali membutuhkan usaha manusia untuk membangun set pelatihan, tetapi kemudian mengotomatisasi dan sering mempercepat tugas yang sulit atau tidak mungkin dilakukan.

#### 2.1 KLASIFIKASI DAN REGRESI

Ada dua jenis utama masalah pembelajaran mesin yang diawasi, yang disebut klasifikasi dan regresi.

Dalam klasifikasi, tujuannya adalah untuk memprediksi label kelas, yang merupakan pilihan dari daftar kemungkinan yang telah ditentukan sebelumnya. Dalam Bab 1 kami menggunakan contoh mengklasifikasikan iris menjadi salah satu dari tiga kemungkinan spesies. Klasifikasi kadang-kadang dipisahkan menjadi klasifikasi biner, yang merupakan kasus khusus untuk membedakan antara tepat dua kelas, dan klasifikasi multiclass, yaitu klasifikasi antara lebih dari dua kelas. Anda dapat menganggap klasifikasi biner sebagai mencoba menjawab pertanyaan ya/tidak. Mengklasifikasikan email sebagai spam atau bukan spam adalah contoh masalah klasifikasi biner. Dalam tugas klasifikasi biner ini, pertanyaan ya/tidak yang diajukan adalah "Apakah ini email spam?"



Dalam klasifikasi biner kita sering berbicara tentang satu kelas menjadi kelas positif dan kelas lainnya menjadi kelas negatif. Di sini, positif tidak merepresentasikan memiliki manfaat atau nilai, melainkan apa objek kajiannya. Jadi, ketika mencari spam, "positif" bisa berarti kelas spam. Manakah dari dua kelas yang disebut positif sering merupakan masalah subjektif, dan khusus untuk domain.

Contoh iris, di sisi lain, adalah contoh dari masalah klasifikasi multiclass. Contoh lain adalah memprediksi bahasa apa yang digunakan situs web dari teks di situs web. Kelas-kelas di sini akan menjadi daftar kemungkinan bahasa yang telah ditentukan sebelumnya.

Untuk tugas regresi, tujuannya adalah untuk memprediksi angka kontinu, atau angka floating-point dalam istilah pemrograman (atau bilangan real dalam istilah matematika). Memprediksi pendapatan tahunan seseorang dari pendidikan, usia, dan tempat tinggal

mereka adalah contoh tugas regresi. Saat memprediksi pendapatan, nilai prediksi adalah jumlah, dan dapat berupa angka apa pun dalam rentang tertentu. Contoh lain dari tugas regresi adalah memprediksi hasil pertanian jagung yang diberikan atribut seperti hasil sebelumnya, cuaca, dan jumlah karyawan yang bekerja di pertanian. Hasil lagi bisa menjadi angka arbitrer.

Cara mudah untuk membedakan antara tugas klasifikasi dan regresi adalah dengan menanyakan apakah ada semacam kontinuitas dalam output. Jika ada kontinuitas antara hasil yang mungkin, maka masalahnya adalah masalah regresi. Pikirkan tentang memprediksi pendapatan tahunan. Ada kontinuitas yang jelas dalam output. Apakah seseorang membuat Rp 600.000.000 atau Rp 600.015.000 setahun tidak membuat perbedaan yang nyata, meskipun ini adalah jumlah uang yang berbeda; jika algoritme kami memprediksi Rp 599.985.000 atau Rp 600.015.000 padahal seharusnya memperkirakan Rp 600.000.000, kami tidak terlalu mempermasalahkannya.

Sebaliknya, untuk tugas mengenali bahasa situs web (yang merupakan masalah klasifikasi), tidak ada masalah derajat. Sebuah situs web dalam satu bahasa, atau dalam bahasa lain. Tidak ada kesinambungan antar bahasa, dan tidak ada bahasa antara Inggris dan Prancis.

## **2.2 GENERALISASI, OVERRFITTIN, DAN UNDERFITTING**

Dalam pembelajaran terawasi, kami ingin membangun model pada data pelatihan dan kemudian dapat membuat prediksi akurat pada data baru yang tidak terlihat yang memiliki karakteristik yang sama dengan set pelatihan yang kami gunakan. Jika sebuah model mampu membuat prediksi yang akurat pada data yang tidak terlihat, kita katakan model tersebut mampu menggeneralisasi dari training set ke test set. Kami ingin membangun model yang mampu menggeneralisasi seakurat mungkin.

Biasanya kami membangun model sedemikian rupa sehingga dapat membuat prediksi yang akurat pada set pelatihan. Jika set pelatihan dan tes memiliki cukup kesamaan, kami berharap model juga akurat pada set tes. Namun, ada beberapa kasus di mana ini bisa salah. Misalkan, jika kita membiarkan diri kita membangun model yang sangat kompleks, kita selalu bisa seakurat yang kita inginkan di set pelatihan.

Mari kita lihat contoh yang dibuat-buat untuk mengilustrasikan hal ini. Katakanlah seorang ilmuwan data pemula ingin memprediksi apakah seorang pelanggan akan membeli perahu, dengan catatan pembeli perahu sebelumnya dan pelanggan yang kita kenal tidak tertarik untuk membeli perahu.<sup>2</sup> Tujuannya adalah untuk mengirimkan email promosi kepada orang-orang yang cenderung benar-benar melakukan pembelian, tetapi tidak mengganggu pelanggan yang tidak akan tertarik.

Misalkan kita memiliki catatan pelanggan yang ditunjukkan pada Tabel 2.1.

**Tabel 2.1** Contoh data tentang pelanggan

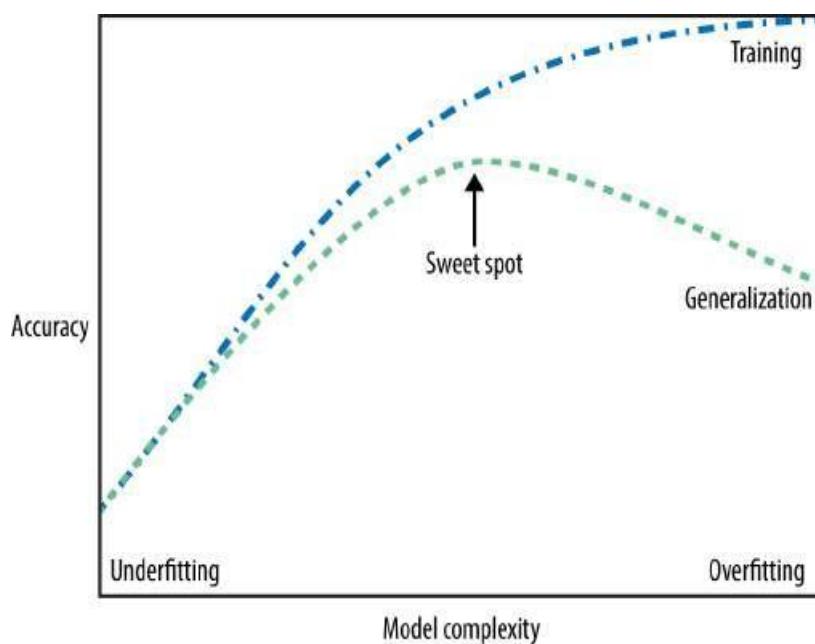
Usia	Jumlah mobil yang dimiliki	Memiliki rumah	Jumlah anak	Status pernikahan	Memiliki seekor anjing	Membeli perahu
66	1	Ya	2	janda	tidak	Ya
52	2	Ya	3	telah menikah	tidak	Ya
22	0	tidak	0	telah menikah	Ya	tidak
25	1	tidak	1	Lajang	tidak	tidak
44	0	tidak	2	Cerai	Ya	tidak
39	1	Ya	2	telah menikah	Ya	tidak
26	1	tidak	2	Lajang	tidak	tidak
40	3	Ya	1	telah menikah	Ya	tidak
53	2	Ya	2	Cerai	tidak	Ya
64	2	Ya	3	Cerai	tidak	tidak
58	2	Ya	2	telah menikah	Ya	Ya
33	1	tidak	1	Lajang	tidak	tidak

Setelah melihat data beberapa saat, ilmuwan data pemula kami membuat aturan berikut: "Jika pelanggan berusia lebih dari 45 tahun, dan memiliki kurang dari 3 anak atau belum bercerai, maka mereka ingin membeli perahu." Ketika ditanya seberapa baik aturan ini, ilmuwan data kami menjawab, "Ini 100 persen akurat!" Dan memang, pada data yang ada di tabel, aturannya sangat akurat. Ada banyak kemungkinan aturan yang dapat kami buat yang akan menjelaskan dengan sempurna jika seseorang dalam kumpulan data ini ingin membeli perahu. Tidak ada usia yang muncul dua kali dalam data, jadi bisa dikatakan orang yang berusia 66, 52, 53, atau 58 tahun ingin membeli perahu, sedangkan yang lainnya tidak. Meskipun kita dapat membuat banyak aturan yang bekerja dengan baik pada data ini, ingatlah bahwa kita tidak tertarik untuk membuat prediksi untuk kumpulan data ini; kita sudah tahu jawaban untuk pelanggan ini. Kami ingin tahu apakah pelanggan baru kemungkinan besar akan membeli kapal. Oleh karena itu kami ingin menemukan aturan yang akan bekerja dengan baik untuk pelanggan baru, dan mencapai akurasi 100 persen pada set pelatihan tidak membantu kami di sana. Kami mungkin tidak berharap bahwa aturan yang dibuat oleh ilmuwan data kami akan bekerja dengan sangat baik pada pelanggan baru. Tampaknya terlalu rumit, dan didukung oleh data yang sangat sedikit. Misalnya, bagian "atau tidak diceraikan" dari aturan bergantung pada satu pelanggan.

Satu-satunya ukuran apakah suatu algoritma akan berkinerja baik pada data baru adalah evaluasi pada set pengujian. Namun, secara intuitif<sup>3</sup> kami mengharapkan model sederhana untuk digeneralisasi lebih baik ke data baru. Jika aturannya adalah "Orang yang berusia lebih dari 50 tahun ingin membeli perahu," dan ini akan menjelaskan perilaku semua pelanggan, kami akan lebih mempercayainya daripada aturan yang melibatkan anak-anak dan status perkawinan selain usia. Oleh karena itu, kami selalu ingin menemukan model yang paling sederhana. Membangun model yang terlalu rumit untuk jumlah informasi yang kita miliki, seperti yang dilakukan ilmuwan data pemula, disebut overfitting. Overfitting terjadi

ketika Anda menyesuaikan model terlalu dekat dengan kekhususan set pelatihan dan mendapatkan model yang bekerja dengan baik pada set pelatihan tetapi tidak dapat digeneralisasi ke data baru. Di sisi lain, jika model Anda terlalu sederhana—misalnya, “Setiap orang yang memiliki rumah membeli perahu”—maka Anda mungkin tidak dapat menangkap semua aspek dan variabilitas dalam data, dan model Anda akan melakukannya buruk bahkan di set pelatihan. Memilih model yang terlalu sederhana disebut underfitting.

Semakin kompleks model yang kita izinkan, semakin baik kita dapat memprediksi data pelatihan. Namun, jika model kami menjadi terlalu kompleks, kami mulai terlalu fokus pada setiap titik data individu dalam set pelatihan kami, dan model tidak akan menggeneralisasi dengan baik ke data baru. Ada titik manis di antara yang akan menghasilkan kinerja generalisasi terbaik. Ini adalah model yang ingin kami temukan. Pertukaran antara overfitting dan underfitting diilustrasikan pada Gambar 2.1.



**Gambar 2.1** Pertukaran kompleksitas model dengan pelatihan dan akurasi pengujian

### 2.3 HUBUNGAN KOMPLEKSITAS MODEL DENGAN UKURAN DATASET

Penting untuk dicatat bahwa kompleksitas model terkait erat dengan variasi input yang terdapat dalam set data pelatihan Anda: semakin banyak variasi titik data yang terdapat dalam set data Anda, semakin kompleks model yang dapat Anda gunakan tanpa overfitting. Biasanya, mengumpulkan lebih banyak titik data akan menghasilkan lebih banyak variasi, sehingga kumpulan data yang lebih besar memungkinkan pembuatan model yang lebih kompleks. Namun, hanya menduplikasi titik data yang sama atau mengumpulkan data yang sangat mirip tidak akan membantu.

Kembali ke contoh penjualan perahu, jika kita melihat 10.000 baris lebih data pelanggan, dan semuanya mematuhi aturan “Jika pelanggan berusia lebih dari 45 tahun, dan memiliki anak kurang dari 3 atau belum bercerai, maka mereka ingin beli perahu,” kami akan lebih percaya bahwa ini adalah aturan yang baik daripada ketika aturan itu dikembangkan hanya dengan menggunakan 12 baris pada Tabel 2.1.

Memiliki lebih banyak data dan membangun model yang lebih kompleks dengan tepat seringkali dapat menghasilkan keajaiban untuk tugas-tugas pembelajaran yang diawasi. Dalam buku ini, kita akan fokus bekerja dengan kumpulan data dengan ukuran tetap. Di dunia nyata, Anda sering memiliki kemampuan untuk memutuskan berapa banyak data yang akan dikumpulkan, yang mungkin lebih bermanfaat daripada mengutak-atik dan menyetel model Anda. Jangan pernah meremehkan kekuatan lebih banyak data.

## 2.4 ALGORITMA PEMBELAJARAN MESIN

Kami sekarang akan meninjau algoritme pembelajaran mesin paling populer dan menjelaskan bagaimana mereka belajar dari data dan bagaimana mereka membuat prediksi. Kami juga akan membahas bagaimana konsep kompleksitas model dimainkan untuk masing-masing model ini, dan memberikan gambaran umum tentang bagaimana setiap algoritma membangun sebuah model. Kami akan memeriksa kekuatan dan kelemahan masing-masing algoritma, dan jenis data apa yang paling baik untuk diterapkan. Kami juga akan menjelaskan arti dari parameter dan opsi yang paling penting.<sup>4</sup> Banyak algoritma memiliki klasifikasi dan varian regresi, dan kami akan menjelaskan keduanya.

Tidak perlu membaca deskripsi setiap algoritme secara mendetail, tetapi memahami model akan memberi Anda perasaan yang lebih baik tentang berbagai cara kerja algoritme pembelajaran mesin. Bab ini juga dapat digunakan sebagai panduan referensi, dan Anda dapat kembali ke sana ketika Anda tidak yakin tentang cara kerja salah satu algoritme.

## 2.5 BEBERAPA CONTOH DATASET

Kami akan menggunakan beberapa kumpulan data untuk menggambarkan algoritma yang berbeda. Beberapa kumpulan data akan berukuran kecil dan sintetis (artinya dibuat-buat), dirancang untuk menyoroti aspek-aspek tertentu dari algoritme. Kumpulan data lain akan menjadi contoh dunia nyata yang besar.

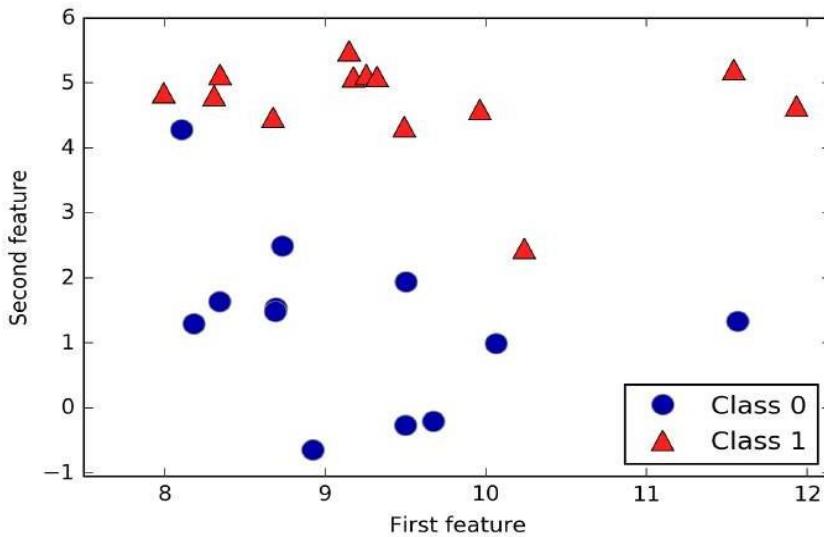
Contoh dataset klasifikasi dua kelas sintetis adalah dataset forge, yang memiliki dua fitur. Kode berikut membuat plot sebar (Gambar 2.2) yang memvisualisasikan semua titik data dalam kumpulan data ini. Plot memiliki ciri pertama pada sumbu x dan ciri kedua pada sumbu y. Seperti yang selalu terjadi dalam plot pencar, setiap titik data direpresentasikan sebagai satu titik. Warna dan bentuk titik menunjukkan kelasnya:

**In[2]:**

```
# generate dataset
X, y = mglearn.datasets.make_forge()
# plot dataset
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")
print("X.shape: {}".format(X.shape))
```

**Out[2]:**

```
X.shape: (26, 2)
```

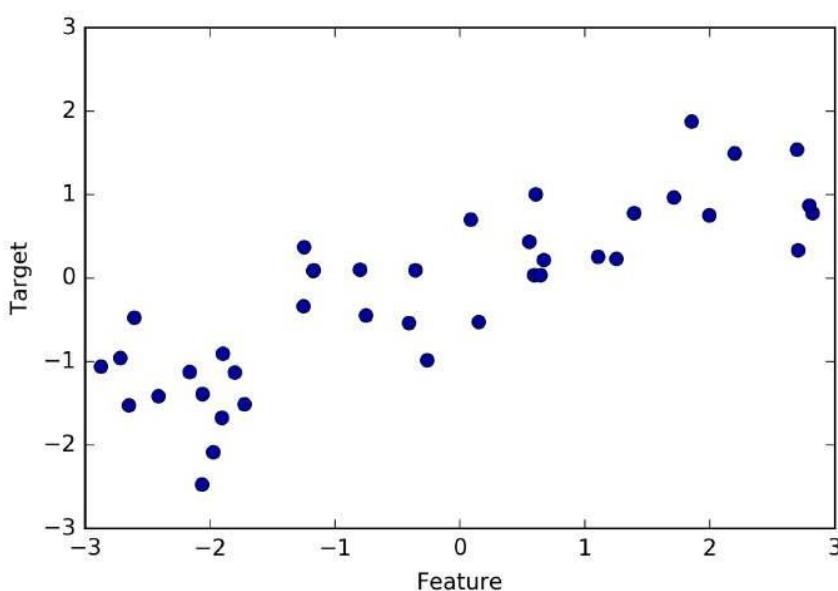


**Gambar 2.2** Plot pencar dari kumpulan data tempa

Seperti yang Anda lihat dari X.shape, dataset ini terdiri dari 26 titik data, dengan 2 fitur. Untuk mengilustrasikan algoritma regresi, kita akan menggunakan dataset gelombang sintetis. Dataset gelombang memiliki fitur input tunggal dan variabel target berkelanjutan (atau respons) yang ingin kita modelkan. Plot yang dibuat di sini (Gambar 2.3) menunjukkan fitur tunggal pada sumbu x dan target regresi (output) pada sumbu y:

**In[3]:**

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Feature")
plt.ylabel("Target")
```



**Gambar 2.3** Plot kumpulan data gelombang, dengan sumbu x menunjukkan fitur dan sumbu y menunjukkan target regresi

Kami menggunakan kumpulan data berdimensi rendah yang sangat sederhana ini karena kami dapat dengan mudah memvisualisasikannya—halaman tercetak memiliki dua dimensi, jadi data dengan lebih dari dua fitur sulit untuk ditampilkan. Intuisi apa pun yang diturunkan dari kumpulan data dengan sedikit fitur (juga disebut kumpulan data berdimensi rendah) mungkin tidak berlaku dalam kumpulan data dengan banyak fitur (kumpulan data berdimensi tinggi). Selama Anda mengingatnya, memeriksa algoritme pada kumpulan data dimensi rendah bisa sangat bermanfaat.

Kami akan melengkapi kumpulan data sintetis kecil ini dengan dua kumpulan data dunia nyata yang disertakan dalam scikit-learn. Salah satunya adalah dataset Kanker Payudara Wisconsin (singkatnya kanker), yang mencatat pengukuran klinis tumor kanker payudara. Setiap tumor diberi label sebagai "jinak" (untuk tumor yang tidak berbahaya) atau "ganous" (untuk tumor kanker), dan tugasnya adalah belajar untuk memprediksi apakah tumor ganous berdasarkan pengukuran jaringan.

Data dapat dimuat menggunakan fungsi `load_breast_cancer` dari scikit-learn:

**In[4]:**

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
```

**Out[4]:**

```
cancer.keys():
dict_keys(['feature_names', 'data', 'DESCR', 'target', 'target_names'])
```



Kumpulan data yang termasuk dalam scikit-learn biasanya disimpan sebagai objek Bunch, yang berisi beberapa informasi tentang kumpulan data serta data aktual. Yang perlu Anda ketahui tentang objek Bunch adalah bahwa objek tersebut berperilaku seperti kamus, dengan manfaat tambahan bahwa Anda dapat mengakses nilai menggunakan titik (seperti pada `bunch.key` alih-alih `bunch['key']`).

Dataset terdiri dari 569 titik data, dengan masing-masing 30 fitur:

**In[5]:**

```
print("Shape of cancer data: {}".format(cancer.data.shape))
```

**Out[5]:**

```
Shape of cancer data: (569, 30)
```

Dari 569 titik data ini, 212 dilabeli sebagai ganous dan 357 sebagai jinak:

**In[6]:**

```
print("Sample counts per class:\n{}".format(
    {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
```

**Out[6]:**

```
Sample counts per class:
{'benign': 357, 'malignant': 212}
```

Untuk mendapatkan deskripsi makna semantik dari setiap fitur, kita dapat melihat atribut feature\_names :

**In[7]:**

```
print("Feature names:\n{}".format(cancer.feature_names))
```

**Out[7]:**

```
Feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

Anda dapat mengetahui lebih lanjut tentang data dengan membaca cancer.DESCR jika Anda tertarik.

Kami juga akan menggunakan dataset regresi dunia nyata, dataset Boston Housing. Tugas yang terkait dengan kumpulan data ini adalah untuk memprediksi nilai median rumah di beberapa lingkungan Boston pada tahun 1970-an, menggunakan informasi seperti tingkat kejahatan, kedekatan dengan Sungai Charles, aksesibilitas jalan raya, dan sebagainya. Dataset berisi 506 titik data, dijelaskan oleh 13 fitur:

**In[8]:**

```
from sklearn.datasets import load_boston
boston = load_boston()
print("Data shape: {}".format(boston.data.shape))
```

**Out[8]:**

```
Data shape: (506, 13)
```

Sekali lagi, Anda bisa mendapatkan informasi lebih lanjut tentang dataset dengan membaca atribut DESCR dari boston. Untuk tujuan kami di sini, kami sebenarnya akan memperluas kumpulan data ini dengan tidak hanya mempertimbangkan 13 pengukuran ini sebagai fitur input, tetapi juga melihat semua produk (juga disebut interaksi) antar fitur. Dengan kata lain, kita tidak hanya akan mempertimbangkan tingkat kejahatan dan aksesibilitas jalan raya sebagai fitur, tetapi juga produk dari tingkat kejahatan dan aksesibilitas jalan raya. Menyertakan fitur turunan seperti ini disebut rekayasa fitur, yang akan kita bahas

lebih detail di Bab 4. Dataset turunan ini dapat dimuat menggunakan fungsi `load_extended_boston`:

**In[9]:**

```
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
```

**Out[9]:**

```
X.shape: (506, 104)
```

104 fitur yang dihasilkan adalah 13 fitur asli bersama dengan 91 kemungkinan kombinasi dua fitur dalam 13.

Kami akan menggunakan kumpulan data ini untuk menjelaskan dan mengilustrasikan properti dari berbagai algoritme pembelajaran mesin. Tetapi untuk saat ini, mari kita membahas algoritma itu sendiri. Pertama, kita akan meninjau kembali algoritma k-nearest neighbor (k-NN) yang kita lihat di bab sebelumnya.

## 2.6 K-NEAREST NEIGHBORS

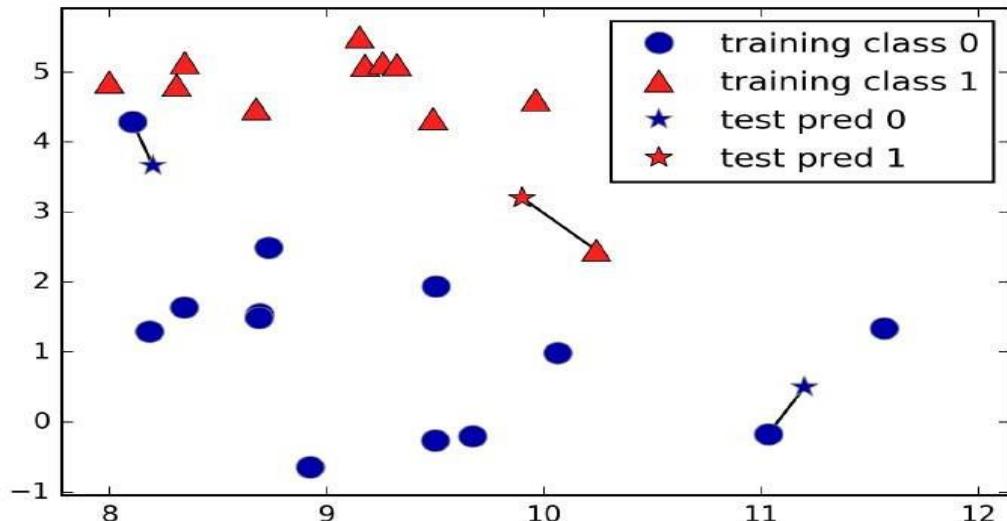
Algoritma k-NN bisa dibilang merupakan algoritma pembelajaran mesin yang paling sederhana. Membangun model hanya terdiri dari menyimpan dataset pelatihan. Untuk membuat prediksi untuk titik data baru, algoritme menemukan titik data terdekat dalam set data pelatihan—"tetangga terdekatnya".

### klasifikasi k-Neighbors

Dalam versi yang paling sederhana, algoritma k-NN hanya mempertimbangkan tepat satu tetangga terdekat, yang merupakan titik data pelatihan terdekat dengan titik yang ingin kita prediksi. Prediksinya kemudian hanyalah output yang diketahui untuk titik pelatihan ini. Gambar 2.4 mengilustrasikan hal ini untuk kasus klasifikasi pada kumpulan data tempa:

**In[10]:**

```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```

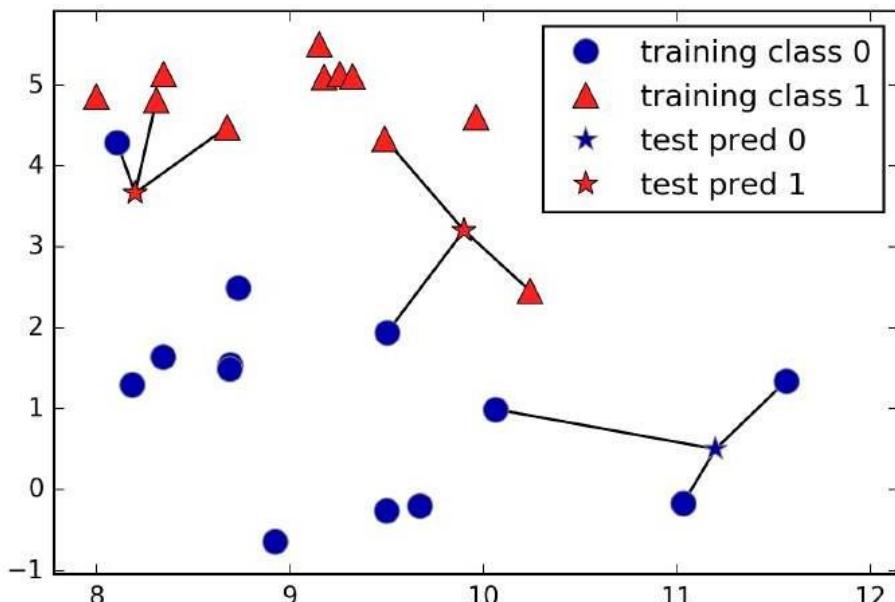


**Gambar 2.4** Prediksi yang dibuat oleh model *one-nearest-neighbor* pada dataset *forge*

Di sini, kami menambahkan tiga titik data baru, yang ditampilkan sebagai bintang. Untuk masing-masing dari mereka, kami menandai titik terdekat di set pelatihan. Prediksi algoritma *one-nearest-neighbor* adalah label dari titik tersebut (ditunjukkan dengan warna salib). Alih-alih hanya mempertimbangkan tetangga terdekat, kita juga dapat mempertimbangkan bilangan arbitrer,  $k$ , dari tetangga. Dari sinilah nama algoritma  $k$ -nearest neighbor berasal. Saat mempertimbangkan lebih dari satu tetangga, kami menggunakan voting untuk menetapkan label. Ini berarti bahwa untuk setiap titik uji, kami menghitung berapa banyak tetangga yang termasuk dalam kelas 0 dan berapa banyak tetangga yang termasuk dalam kelas 1. Kami kemudian menetapkan kelas yang lebih sering: dengan kata lain, kelas mayoritas di antara  $k$ -tetangga terdekat. Contoh berikut (Gambar 2.5) menggunakan tiga tetangga terdekat:

In[11]:

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```



**Gambar 2.5** Prediksi yang dibuat oleh model *Three-neighbors-nearest* pada kumpulan data tempa

Sekali lagi, prediksi ditampilkan sebagai warna salib. Anda dapat melihat bahwa prediksi untuk titik data baru di kiri atas tidak sama dengan prediksi ketika kita hanya menggunakan satu tetangga.

Meskipun ilustrasi ini untuk masalah klasifikasi biner, metode ini dapat diterapkan pada kumpulan data dengan sejumlah kelas. Untuk lebih banyak kelas, kami menghitung berapa banyak tetangga milik setiap kelas dan sekali lagi memprediksi kelas yang paling umum.

Sekarang mari kita lihat bagaimana kita dapat menerapkan algoritma  $k$ -nearest neighbor menggunakan scikit-learn. Pertama, kami membagi data kami menjadi satu set

pelatihan dan pengujian sehingga kami dapat mengevaluasi kinerja generalisasi, seperti yang dibahas dalam Bab 1:

**In[12]:**

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Selanjutnya, kita mengimpor dan membuat instance kelas. Ini adalah saat kita dapat mengatur parameter, seperti jumlah tetangga yang akan digunakan. Di sini, kami mengurnya ke 3:

**In[13]:**

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

Sekarang, kami menyesuaikan pengklasifikasi menggunakan set pelatihan. Untuk KNeighborsClassifier ini berarti menyimpan dataset, sehingga kita dapat menghitung tetangga selama prediksi:

**In[14]:**

```
clf.fit(X_train, y_train)
```

Untuk membuat prediksi pada data uji, kami menyebutnya metode prediksi. Untuk setiap titik data di set pengujian, ini menghitung tetangga terdekatnya di set pelatihan dan menemukan kelas yang paling umum di antara ini:

**In[15]:**

```
print("Test set predictions: {}".format(clf.predict(X_test)))
```

**Out[15]:**

```
Test set predictions: [1 0 1 0 1 0 0]
```

Untuk mengevaluasi seberapa baik model kami digeneralisasi, kami dapat memanggil metode skor dengan data uji bersama dengan label uji:

**In[16]:**

```
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

**Out[16]:**

```
Test set accuracy: 0.86
```

Kami melihat bahwa model kami sekitar 86% akurat, artinya model memprediksi kelas dengan benar untuk 86% sampel dalam kumpulan data uji.

### Menganalisis KNeighborsClassifier

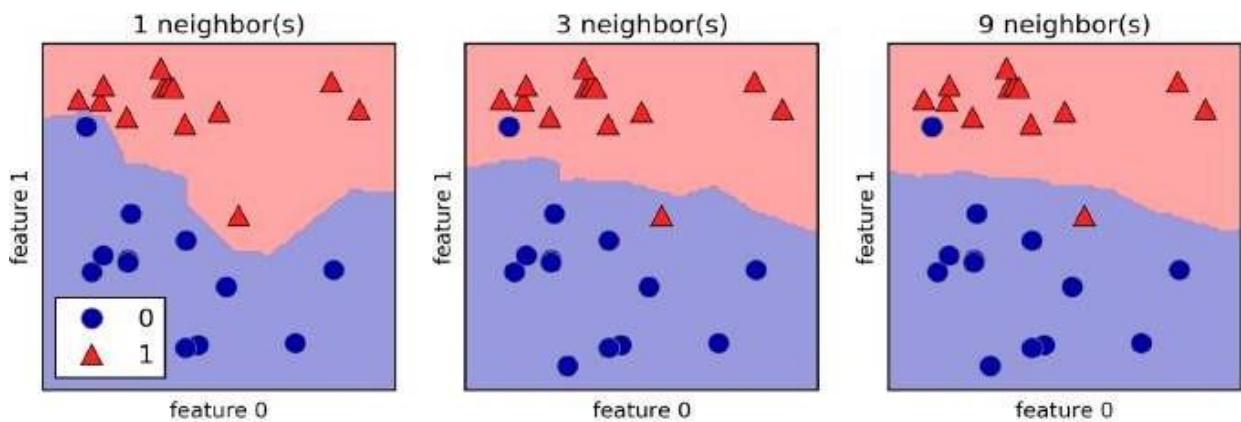
Untuk kumpulan data dua dimensi, kita juga dapat mengilustrasikan prediksi untuk semua titik uji yang mungkin dalam bidang xy. Kami mewarnai pesawat sesuai dengan kelas yang akan ditugaskan ke titik di wilayah ini. Ini memungkinkan kita melihat batas keputusan, yang merupakan pembagian antara tempat algoritme menetapkan kelas 0 versus di mana ia menetapkan kelas 1.

Kode berikut menghasilkan visualisasi batas keputusan untuk satu, tiga, dan sembilan tetangga yang ditunjukkan pada Gambar 2.6:

In[17]:

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # the fit method returns the object self, so we can instantiate
    # and fit in one line
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} neighbor(s)".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
    axes[0].legend(loc=3)
```



**Gambar 2.6** Batas keputusan yang dibuat oleh model tetangga terdekat untuk nilai n\_tetangga yang berbeda

Seperti yang Anda lihat di sebelah kiri pada gambar, menggunakan tetangga tunggal menghasilkan batas keputusan yang mengikuti data pelatihan dengan cermat. Mempertimbangkan semakin banyak tetangga mengarah ke batas keputusan yang lebih halus. Batas yang lebih halus sesuai dengan model yang lebih sederhana. Dengan kata lain, menggunakan beberapa tetangga sesuai dengan kompleksitas model yang tinggi (seperti yang ditunjukkan di sisi kanan Gambar 2.6), dan menggunakan banyak tetangga sesuai dengan kompleksitas model yang rendah (seperti yang ditunjukkan di sisi kiri Gambar 2.6). Jika Anda mempertimbangkan kasus ekstrim di mana jumlah tetangga adalah jumlah semua titik data dalam set pelatihan, setiap titik uji akan memiliki tetangga yang persis sama (semua titik pelatihan) dan semua prediksi akan sama: kelas yang paling sering di set pelatihan.

Mari kita selidiki apakah kita dapat mengkonfirmasi hubungan antara kompleksitas model dan generalisasi yang telah kita bahas sebelumnya. Kami akan melakukan ini pada dataset Kanker Payudara dunia nyata. Kita mulai dengan membagi dataset menjadi training dan test set. Kemudian kami mengevaluasi pelatihan dan pengujian kinerja set dengan jumlah tetangga yang berbeda. Hasilnya ditunjukkan pada Gambar 2.7:

In[18]:

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

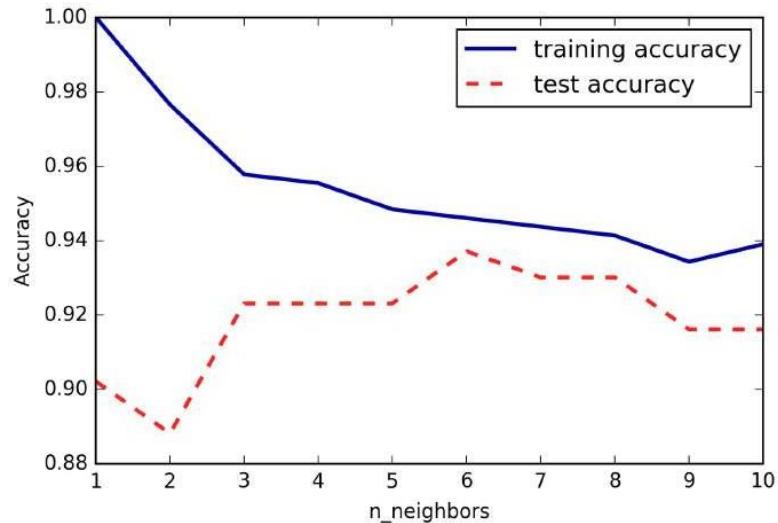
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # build the model
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(clf.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

Plot menunjukkan akurasi set pelatihan dan pengujian pada sumbu y terhadap pengaturan n\_tetangga pada sumbu x. Sementara plot dunia nyata jarang sangat mulus, kita masih dapat mengenali beberapa karakteristik overfitting dan underfitting (perhatikan bahwa karena mempertimbangkan lebih sedikit tetangga sesuai dengan model yang lebih kompleks, plot dibalik secara horizontal relatif terhadap ilustrasi pada Gambar 2.7). Mengingat satu tetangga terdekat, prediksi pada set pelatihan sempurna. Tetapi ketika lebih banyak tetangga dipertimbangkan, model menjadi lebih sederhana dan akurasi pelatihan turun. Akurasi test set untuk menggunakan satu tetangga lebih rendah daripada ketika menggunakan lebih banyak tetangga, menunjukkan bahwa menggunakan satu tetangga terdekat mengarah ke model yang terlalu kompleks.

Di sisi lain, ketika mempertimbangkan 10 tetangga, modelnya terlalu sederhana dan kinerjanya bahkan lebih buruk. Performa terbaik ada di suatu tempat di tengah, menggunakan sekitar enam tetangga. Namun, ada baiknya untuk mengingat skala plot. Performa terburuk adalah akurasi sekitar 88%, yang mungkin masih dapat diterima.

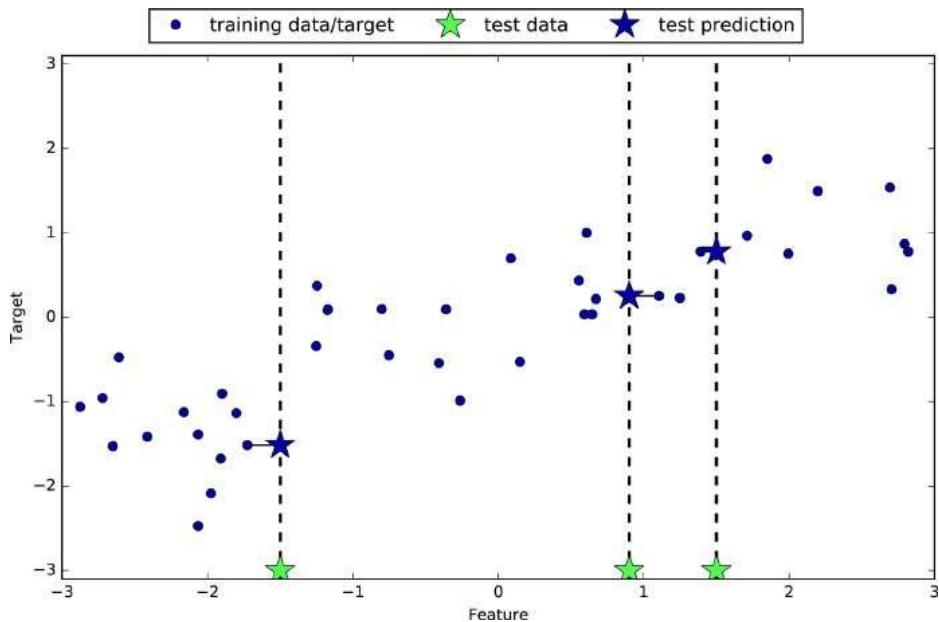


**Gambar 2.7** Perbandingan akurasi pelatihan dan pengujian sebagai fungsi  $n_{\text{tetangga}}$  regresi k-tetangga

Ada juga varian regresi dari algoritma k-nearest neighbor. Sekali lagi, mari kita mulai dengan menggunakan satu tetangga terdekat, kali ini menggunakan dataset gelombang. Kami telah menambahkan tiga titik data pengujian sebagai bintang hijau pada sumbu x. Prediksi menggunakan satu tetangga hanyalah nilai target dari tetangga terdekat. Ini ditunjukkan sebagai bintang biru pada Gambar 2.8:

**In[19]:**

```
mglearn.plots.plot_knn_regression(n_neighbors=1)
```

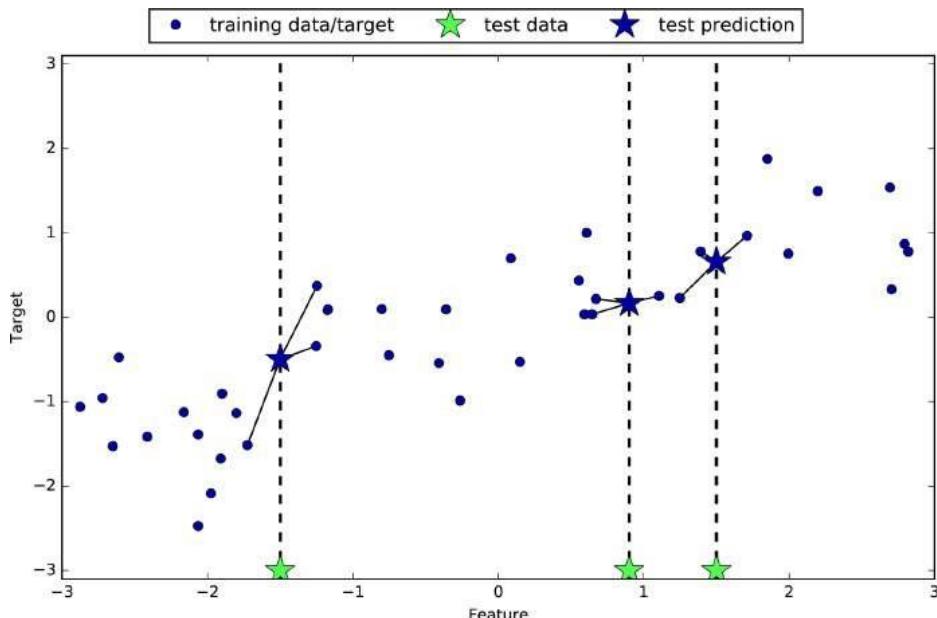


**Gambar 2.8** Prediksi yang dibuat oleh regresi *one-nearest-neighbor* pada kumpulan data gelombang

Sekali lagi, kita dapat menggunakan lebih dari *one-nearest-neighbor* untuk regresi. Saat menggunakan beberapa tetangga terdekat, prediksi adalah rata-rata, atau rata-rata, dari tetangga yang relevan (Gambar 2.9):

In[20]:

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```



**Gambar 2.9** Prediksi yang dibuat oleh regresi *three-nearest-neighbor* pada dataset gelombang

Algoritma k-nearest neighbor untuk regresi diimplementasikan di kelas KNeighborsRegressor di scikit-learn. Ini digunakan mirip dengan KNeighborsClassifier:

In[21]:

```
from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)

# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)
# fit the model using the training data and training targets
reg.fit(X_train, y_train)
```

Sekarang kita dapat membuat prediksi pada set tes:

In[22]:

```
print("Test set predictions:\n{}".format(reg.predict(X_test)))
```

Out[22]:

```
Test set predictions:
[-0.054  0.357  1.137 -1.894 -1.139 -1.631  0.357  0.912 -0.447 -1.139]
```

Kami juga dapat mengevaluasi model menggunakan metode skor, yang untuk regressor mengembalikan skor R<sup>2</sup>. Skor R<sup>2</sup>, juga dikenal sebagai koefisien determinasi, adalah ukuran kebaikan prediksi untuk model regresi, dan menghasilkan skor antara 0 dan 1. Nilai 1 sesuai dengan prediksi sempurna, dan nilai 0 sesuai dengan model konstan yang hanya memprediksi rata-rata respons set pelatihan, y\_train:

**In[23]:**

```
print("Test set R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

**Out[23]:**

```
Test set R^2: 0.83
```

Di sini, skornya adalah 0,83, yang menunjukkan model fit yang relatif baik.

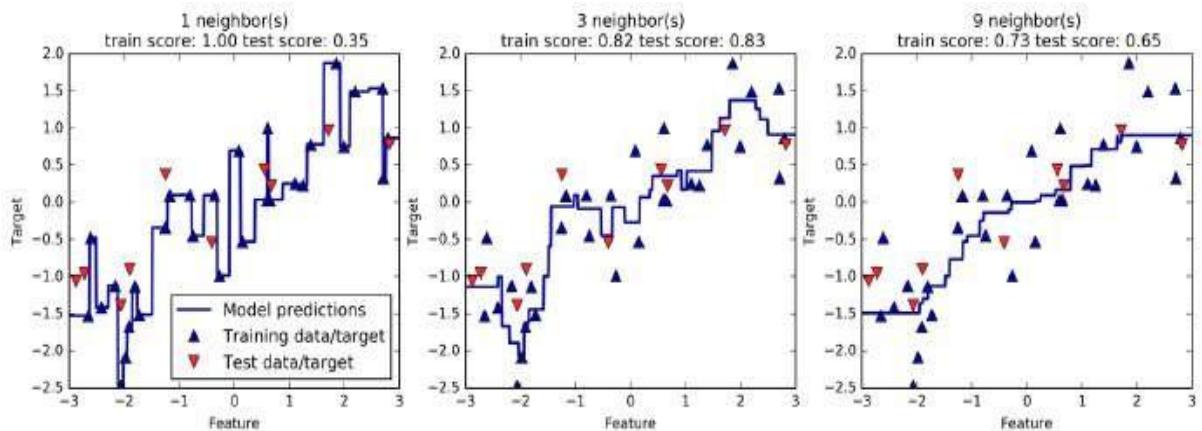
### Menganalisis KNeighborsRegressor

Untuk dataset satu dimensi, kita dapat melihat seperti apa prediksi untuk semua nilai fitur yang mungkin (Gambar 2.10). Untuk melakukan ini, kami membuat kumpulan data uji yang terdiri dari banyak titik di telepon:

**In[24]:**

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# create 1,000 data points, evenly spaced between -3 and 3
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # make predictions using 1, 3, or 9 neighbors
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mlearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mlearn.cm2(1), markersize=8)

    ax.set_title(
        "{} neighbor(s)\ntrain score: {:.2f} test score: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train),
            reg.score(X_test, y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")
    axes[0].legend(["Model predictions", "Training data/target",
                    "Test data/target"], loc="best")
```



**Gambar 2.10** Membandingkan prediksi yang dibuat oleh regresi tetangga terdekat untuk nilai n\_tetangga yang berbeda

Seperti yang dapat kita lihat dari plot, hanya dengan menggunakan satu tetangga, setiap titik dalam set pelatihan memiliki pengaruh yang jelas pada prediksi, dan nilai prediksi melewati semua titik data. Ini mengarah pada prediksi yang sangat tidak stabil. Mempertimbangkan lebih banyak tetangga mengarah ke prediksi yang lebih mulus, tetapi ini juga tidak sesuai dengan data pelatihan.

### Kekuatan, kelemahan, dan parameter

Pada prinsipnya, ada dua parameter penting untuk pengklasifikasi KNeighbors: jumlah tetangga dan cara Anda mengukur jarak antar titik data. Dalam praktiknya, menggunakan sejumlah kecil tetangga seperti tiga atau lima sering kali berfungsi dengan baik, tetapi Anda tentu harus menyesuaikan parameter ini. Memilih ukuran jarak yang tepat agak di luar cakupan buku ini. Secara default, jarak Euclidean digunakan, yang bekerja dengan baik di banyak pengaturan.

Salah satu kelebihan k-NN adalah modelnya sangat mudah dipahami, dan seringkali memberikan kinerja yang wajar tanpa banyak penyesuaian. Menggunakan algoritma ini adalah metode dasar yang baik untuk dicoba sebelum mempertimbangkan teknik yang lebih maju. Membangun model tetangga terdekat biasanya sangat cepat, tetapi ketika set pelatihan Anda sangat besar (baik dalam jumlah fitur atau jumlah sampel) prediksi bisa lambat. Saat menggunakan algoritme k-NN, penting untuk melakukan praproses data Anda (lihat Bab 3). Pendekatan ini sering kali tidak berkinerja baik pada kumpulan data dengan banyak fitur (ratusan atau lebih), dan sangat buruk dengan kumpulan data di mana sebagian besar fitur adalah 0 sebagian besar waktu (disebut kumpulan data jarang).

Jadi, walaupun algoritma k-neighbors terdekat mudah dipahami, algoritma ini tidak sering digunakan dalam praktik, karena prediksi yang lambat dan ketidakmampuannya menangani banyak fitur. Metode yang kita bahas selanjutnya tidak memiliki kekurangan ini.

## 2.7 MODEL LINIER

Model linier adalah kelas model yang banyak digunakan dalam praktik dan telah dipelajari secara ekstensif dalam beberapa dekade terakhir, dengan akhirnya kembali lebih dari seratus tahun. Model linier membuat prediksi menggunakan fungsi linier dari fitur input, yang akan kami jelaskan segera.

### Model linier untuk regresi

Untuk regresi, rumus prediksi umum untuk model linier terlihat sebagai berikut:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

Di sini,  $x[0]$  hingga  $x[p]$  menunjukkan fitur (dalam contoh ini, jumlah fitur adalah  $p$ ) dari satu titik data,  $w$  dan  $b$  adalah parameter model yang dipelajari, dan  $\hat{y}$  adalah prediksi model membuat. Untuk kumpulan data dengan satu fitur, ini adalah:

$$\hat{y} = w[0] * x[0] + b$$

yang mungkin Anda ingat dari matematika SMA sebagai persamaan garis. Di sini,  $w[0]$  adalah kemiringan dan  $b$  adalah offset sumbu  $y$ . Untuk fitur lainnya,  $w$  berisi kemiringan di

sepanjang setiap sumbu fitur. Atau, Anda dapat menganggap respons yang diprediksi sebagai jumlah bobot dari fitur input, dengan bobot (yang bisa negatif) diberikan oleh entri  $w$ .

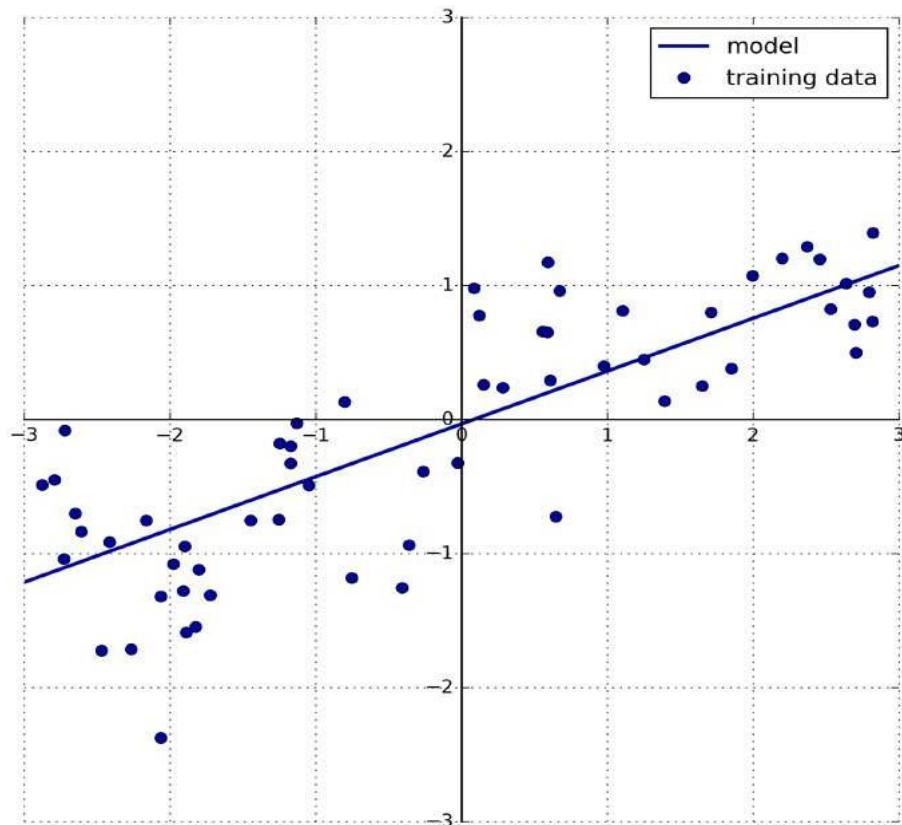
Mencoba mempelajari parameter  $w[0]$  dan  $b$  pada dataset gelombang satu dimensi kami mungkin mengarah ke baris berikut (lihat Gambar 2-11):

**In[25]:**

```
mlearn.plots.plot_linear_regression_wave()
```

**Out[25]:**

```
w[0]: 0.393906 b: -0.031804
```



**Gambar 2.11** Prediksi model linier pada dataset gelombang

Kami menambahkan tanda silang koordinat ke dalam plot untuk memudahkan memahami garis. Melihat  $w[0]$  kita melihat bahwa kemiringan harus sekitar 0,4, yang dapat kita konfirmasi secara visual di plot. Intersep adalah di mana garis prediksi harus melintasi sumbu y: ini sedikit di bawah nol, yang juga dapat Anda konfirmasikan dalam gambar.

Model linier untuk regresi dapat dicirikan sebagai model regresi yang prediksinya berupa garis untuk fitur tunggal, bidang saat menggunakan dua fitur, atau bidang hiper dalam dimensi yang lebih tinggi (yaitu, saat menggunakan lebih banyak fitur).

Jika Anda membandingkan prediksi yang dibuat dengan garis lurus dengan prediksi yang dibuat oleh KNeighborsRegressor pada Gambar 2-10, menggunakan garis lurus untuk membuat prediksi tampaknya sangat membatasi. Sepertinya semua detail halus dari data hilang. Dalam arti tertentu, ini benar. Ini adalah asumsi yang kuat (dan agak tidak realistik)

bahwa target kami  $y$  adalah kombinasi linier dari fitur. Tetapi melihat data satu dimensi memberikan perspektif yang agak miring. Untuk kumpulan data dengan banyak fitur, model linier bisa sangat kuat. Khususnya, jika Anda memiliki lebih banyak fitur daripada titik data pelatihan, target  $y$  apa pun dapat dimodelkan dengan sempurna (pada set pelatihan) sebagai fungsi linier.

Ada banyak model linier yang berbeda untuk regresi. Perbedaan antara model ini terletak pada bagaimana parameter model  $w$  dan  $b$  dipelajari dari data pelatihan, dan bagaimana kompleksitas model dapat dikontrol. Sekarang kita akan melihat model linier paling populer untuk regresi.

### Regresi linier (alias kuadrat terkecil biasa)

Regresi linier, atau kuadrat terkecil biasa (OLS), adalah metode linier paling sederhana dan paling klasik untuk regresi. Regresi linier menemukan parameter  $w$  dan  $b$  yang meminimalkan kesalahan kuadrat rata-rata antara prediksi dan target regresi yang sebenarnya,  $y$ , pada set pelatihan. Kesalahan kuadrat rata-rata adalah jumlah perbedaan kuadrat antara prediksi dan nilai sebenarnya. Regresi linier tidak memiliki parameter, yang merupakan manfaat, tetapi juga tidak memiliki cara untuk mengontrol kompleksitas model. Berikut adalah kode yang menghasilkan model yang dapat Anda lihat pada Gambar 2.11:

**In[26]:**

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

lr = LinearRegression().fit(X_train, y_train)
```

Parameter "kemiringan" ( $w$ ), juga disebut bobot atau koefisien, disimpan dalam `koefisien_atribut`, sedangkan offset atau intersep ( $b$ ) disimpan dalam atribut `intersep`:

**In[27]:**

```
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
```

**Out[27]:**

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759746
```



Anda mungkin melihat garis bawah trailing yang tampak aneh di akhir `coef_` dan `intercept_`. scikit-learn selalu menyimpan apa pun yang berasal dari data pelatihan dalam atribut yang diakhiri dengan garis bawah tambahan. Yaitu untuk memisahkannya dari parameter yang ditetapkan oleh pengguna.

Atribut `intercept_` selalu berupa angka float tunggal, sedangkan atribut `coef_` adalah array NumPy dengan satu entri per fitur input. Karena kami hanya memiliki fitur input tunggal dalam kumpulan data gelombang, `lr.coef_` hanya memiliki satu entri. Mari kita lihat set pelatihan dan kinerja set pengujian:

In[28]:

```
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

Out[28]:

```
Training set score: 0.67
Test set score: 0.66
```

R2 sekitar 0,66 tidak terlalu bagus, tetapi kita dapat melihat bahwa skor pada set pelatihan dan tes sangat berdekatan. Ini berarti kita cenderung underfitting, bukan overfitting. Untuk dataset satu dimensi ini, ada sedikit bahaya overfitting, karena modelnya sangat sederhana (atau dibatas). Namun, dengan kumpulan data berdimensi lebih tinggi (artinya kumpulan data dengan sejumlah besar fitur), model linier menjadi lebih kuat, dan ada kemungkinan overfitting yang lebih tinggi. Mari kita lihat bagaimana kinerja regresi LinearRe pada dataset yang lebih kompleks, seperti dataset Boston Housing. Ingat bahwa dataset ini memiliki 506 sampel dan 105 fitur turunan. Pertama, kita memuat dataset dan membaginya menjadi training dan test set. Kemudian kita membangun model regresi linier seperti sebelumnya:

In[29]:

```
X, y = mglearn.datasets.load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
```

Saat membandingkan skor set pelatihan dan set tes, kami menemukan bahwa kami memprediksi dengan sangat akurat pada set pelatihan, tetapi R2 pada set tes jauh lebih buruk:

In[30]:

```
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

Out[30]:

```
Training set score: 0.95
Test set score: 0.61
```

Perbedaan antara kinerja pada set pelatihan dan set pengujian ini merupakan tanda yang jelas dari overfitting, dan oleh karena itu kita harus mencoba menemukan model yang memungkinkan kita untuk mengontrol kompleksitas. Salah satu alternatif yang paling umum digunakan untuk regresi linier standar adalah regresi punggungan, yang akan kita bahas selanjutnya.

### Regresi punggungan

Regresi punggungan juga merupakan model linier untuk regresi, sehingga rumus yang digunakan untuk membuat prediksi sama dengan yang digunakan untuk kuadrat terkecil biasa. Namun, dalam regresi punggungan, koefisien ( $w$ ) dipilih tidak hanya agar dapat diprediksi dengan baik pada data pelatihan, tetapi juga agar sesuai dengan batasan tambahan. Kami juga ingin besarnya koefisien menjadi sekecil mungkin; dengan kata lain, semua entri  $w$  harus mendekati nol. Secara intuitif, ini berarti setiap fitur harus memiliki efek sesedikit

mungkin pada hasil (yang berarti memiliki kemiringan kecil), sambil tetap memprediksi dengan baik. Batasan ini adalah contoh dari apa yang disebut regularisasi. Regularisasi berarti secara eksplisit membatasi model untuk menghindari overfitting. Jenis khusus yang digunakan oleh regresi ridge dikenal sebagai regularisasi L2

Regresi Ridge diimplementasikan dalam `linear_model.Ridge`. Mari kita lihat seberapa baik kinerjanya pada kumpulan data Boston Housing yang diperluas:

**In[31]:**

```
from sklearn.linear_model import Ridge

ridge = Ridge().fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))
```

**Out[31]:**

```
Training set score: 0.89
Test set score: 0.75
```

Seperti yang Anda lihat, skor set pelatihan Ridge lebih rendah daripada `LinearRegression`, sedangkan skor set tes lebih tinggi. Hal ini sesuai dengan harapan kami. Dengan regresi linier, kami melakukan overfitting data kami. Ridge adalah model yang lebih terbatas, jadi kami cenderung tidak mengenakan pakaian berlebihan. Model yang kurang kompleks berarti kinerja yang lebih buruk pada set pelatihan, tetapi generalisasi yang lebih baik. Karena kita hanya tertarik pada kinerja generalisasi, kita harus memilih model Ridge daripada model `LinearRegression`.

Model Ridge membuat trade-off antara kesederhanaan model (koefisien mendekati nol) dan kinerjanya pada set pelatihan. Seberapa penting model menempatkan kesederhanaan versus kinerja set pelatihan dapat ditentukan oleh pengguna, menggunakan parameter alfa. Pada contoh sebelumnya, kami menggunakan parameter default `alpha=1.0`. Tidak ada alasan mengapa ini akan memberi kita trade-off terbaik. Pengaturan alpha yang optimal tergantung pada dataset tertentu yang kita gunakan. Meningkatkan koefisien kekuatan alfa untuk bergerak lebih menuju nol, yang menurunkan kinerja set pelatihan tetapi mungkin membantu generalisasi. Sebagai contoh:

**In[32]:**

```
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge10.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge10.score(X_test, y_test)))
```

**Out[32]:**

```
Training set score: 0.79
Test set score: 0.64
```

Penurunan alpha memungkinkan koefisien menjadi kurang dibatasi, yang berarti kita bergerak ke kanan pada Gambar 2.1. Untuk nilai alfa yang sangat kecil, koefisien hampir tidak dibatasi sama sekali, dan kami berakhir dengan model yang menyerupai `LinearRegression`:

In[33]:

```
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge01.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge01.score(X_test, y_test)))
```

Out[33]:

```
Training set score: 0.93
Test set score: 0.77
```

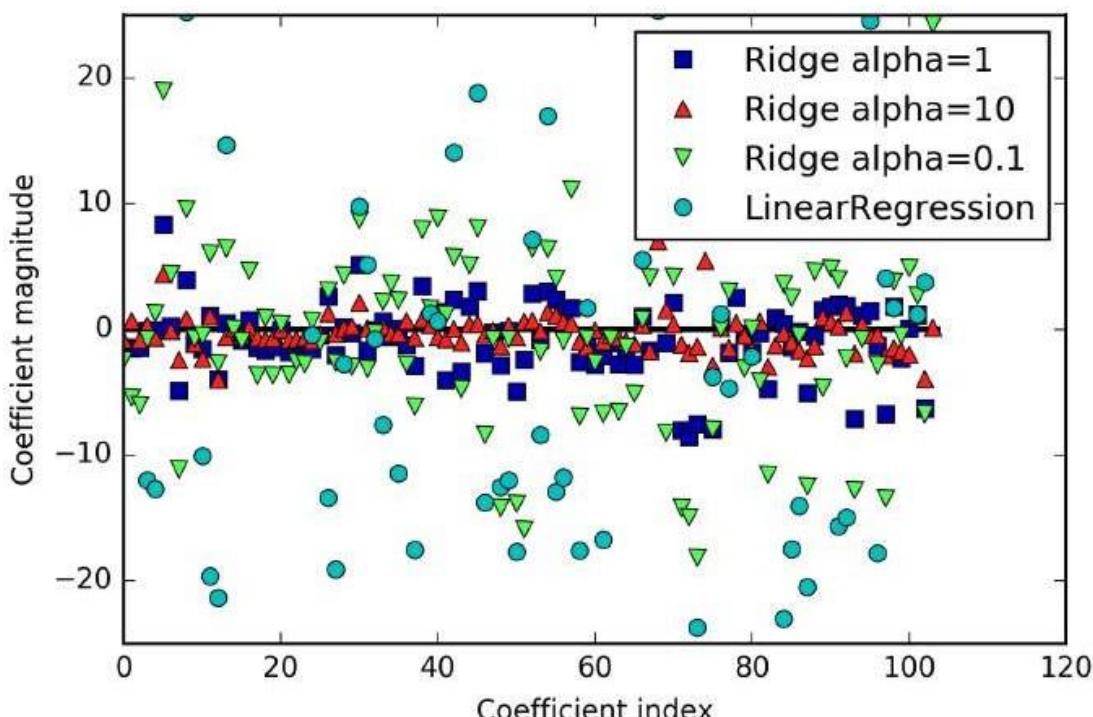
Di sini, alpha=0.1 tampaknya bekerja dengan baik. Kita bisa mencoba mengurangi alpha lebih banyak lagi untuk meningkatkan generalisasi. Untuk saat ini, perhatikan bagaimana parameter alpha sesuai dengan kompleksitas model seperti yang ditunjukkan pada Gambar 2-1. Kami akan membahas metode untuk memilih parameter dengan benar di Bab 5.

Kita juga bisa mendapatkan wawasan yang lebih kualitatif tentang bagaimana parameter alfa mengubah model dengan memeriksa atribut `coef_` model dengan nilai alfa yang berbeda. Alfa yang lebih tinggi berarti model yang lebih terbatas, jadi kami berharap entri `coef_` memiliki magnitudo yang lebih kecil untuk nilai alfa yang tinggi daripada nilai alfa yang rendah. Hal ini ditegaskan dalam plot pada Gambar 2.12:

In[34]:

```
plt.plot(ridge.coef_, 's', label="Ridge alpha=1")
plt.plot(ridge10.coef_, '^', label="Ridge alpha=10")
plt.plot(ridge01.coef_, 'v', label="Ridge alpha=0.1")

plt.plot(lr.coef_, 'o', label="LinearRegression")
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.hlines(0, 0, len(lr.coef_))
plt.ylim(-25, 25)
plt.legend()
```



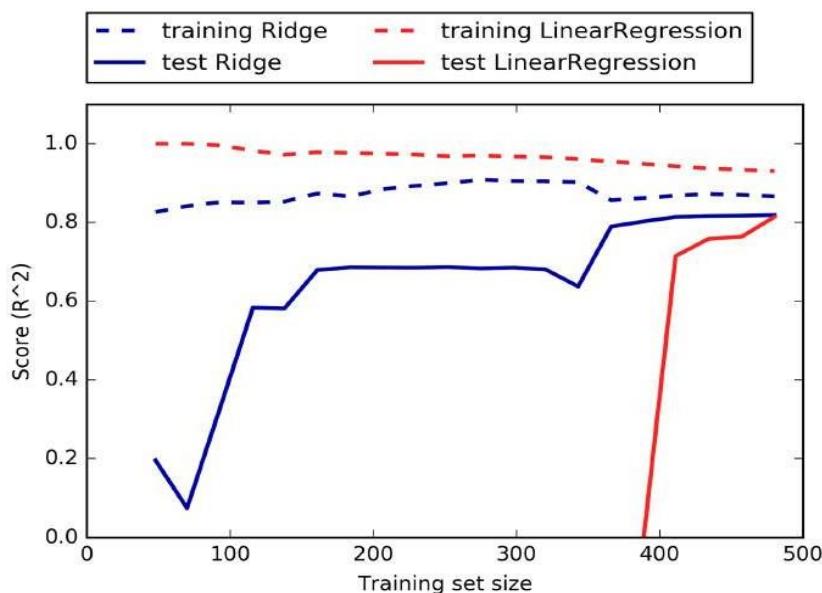
**Gambar 2.12** Membandingkan besaran koefisien untuk regresi ridge dengan nilai regresi alfa dan linier yang berbeda

Di sini, sumbu x menghitung entri dari `coef_`:  $x=0$  menunjukkan koefisien yang terkait dengan fitur pertama,  $x=1$  koefisien yang terkait dengan fitur kedua, dan seterusnya hingga  $x=100$ . Sumbu y menunjukkan nilai numerik dari nilai koefisien yang sesuai. Takeaway utama di sini adalah bahwa untuk  $\alpha=10$ , koefisien sebagian besar antara sekitar -3 dan 3. Koefisien untuk model Ridge dengan  $\alpha=1$  agak lebih besar. Titik-titik yang sesuai dengan  $\alpha=0,1$  masih memiliki magnitudo yang lebih besar, dan banyak titik yang berhubungan dengan regresi linier tanpa regularisasi apa pun (yang akan menjadi  $\alpha=0$ ) begitu besar sehingga berada di luar grafik.

Cara lain untuk memahami pengaruh regularisasi adalah dengan memperbaiki nilai  $\alpha$  tetapi memvariasikan jumlah data pelatihan yang tersedia. Untuk Gambar 2-13, kami membuat subsampel dataset Boston Housing dan mengevaluasi `LinearRegression` dan `Ridge(alpha=1)` pada subset dengan ukuran yang meningkat (plot yang menunjukkan kinerja model sebagai fungsi dari ukuran set data disebut kurva pembelajaran):

**In[35]:**

```
mglearn.plots.plot_ridge_n_samples()
```



**Gambar 2.13** Kurva pembelajaran untuk regresi ridge dan regresi linier pada dataset Boston Housing

Seperti yang diharapkan, skor pelatihan lebih tinggi daripada skor tes untuk semua ukuran kumpulan data, baik untuk regresi ridge maupun linier. Karena ridge diatur, skor pelatihan ridge lebih rendah dari skor pelatihan untuk regresi linier di seluruh papan. Namun, skor tes untuk ridge lebih baik, terutama untuk subset data yang kecil. Untuk kurang dari 400 titik data, regresi linier tidak dapat mempelajari apa pun. Karena semakin banyak data yang tersedia untuk model, kedua model meningkat, dan regresi linier mengejar ridge pada akhirnya. Pelajaran di sini adalah bahwa dengan data pelatihan yang cukup, regularisasi menjadi kurang penting, dan dengan data yang cukup, ridge dan regresi linier akan memiliki

kinerja yang sama (fakta bahwa hal ini terjadi di sini ketika menggunakan dataset penuh hanyalah kebetulan). Aspek lain yang menarik dari Gambar 2-13 adalah penurunan kinerja pelatihan untuk regresi linier. Jika lebih banyak data ditambahkan, model menjadi lebih sulit untuk menyesuaikan, atau mengingat data

### Laso

Sebuah alternatif untuk Ridge untuk mengatur regresi linier adalah Lasso. Seperti halnya regresi ridge, penggunaan laso juga membatasi koefisien agar mendekati nol, tetapi dengan cara yang sedikit berbeda, yang disebut regularisasi L1.8 Konsekuensi dari regularisasi L1 adalah ketika menggunakan laso, beberapa koefisien persis nol. Ini berarti beberapa fitur sepenuhnya diabaikan oleh model. Ini dapat dilihat sebagai bentuk pemilihan fitur otomatis. Memiliki beberapa koefisien yang tepat nol sering kali membuat model lebih mudah untuk diinterpretasikan, dan dapat mengungkapkan fitur paling penting dari model Anda. Mari terapkan laso ke kumpulan data Boston Housing yang diperluas:

**In[36]:**

```
from sklearn.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_ != 0)))
```

**Out[36]:**

```
Training set score: 0.29
Test set score: 0.21
Number of features used: 4
```

Seperti yang Anda lihat, Lasso melakukannya dengan sangat buruk, baik di sesi latihan maupun tes. Ini menunjukkan bahwa kami kurang pas, dan kami menemukan bahwa itu hanya menggunakan 4 dari 105 fitur. Sama halnya dengan Ridge, Lasso juga memiliki parameter regularisasi, alfa, yang mengontrol seberapa kuat koefisien didorong menuju nol. Pada contoh sebelumnya, kami menggunakan default alpha=1.0. Untuk mengurangi underfitting, mari kita coba mengurangi alpha. Ketika kita melakukan ini, kita juga perlu meningkatkan pengaturan default max\_iter (jumlah maksimum iterasi untuk dijalankan):

**In[37]:**

```
# we increase the default setting of "max_iter",
# otherwise the model would warn us that we should increase max_iter.
lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso001.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso001.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso001.coef_ != 0)))
```

**Out[37]:**

```
Training set score: 0.90
Test set score: 0.77
Number of features used: 33
```

Alfa yang lebih rendah memungkinkan kami untuk menyesuaikan model yang lebih kompleks, yang bekerja lebih baik pada data pelatihan dan pengujian. Performanya sedikit

lebih baik daripada menggunakan Ridge, dan kami hanya menggunakan 33 dari 105 fitur. Hal ini membuat model ini berpotensi lebih mudah untuk dipahami.

Namun, jika kami menetapkan alfa terlalu rendah, kami kembali menghapus efek regularisasi dan berakhir dengan overfitting, dengan hasil yang mirip dengan LinearRegression:

**In[38]:**

```
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("Training set score: {:.2f}".format(lasso00001.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso00001.score(X_test, y_test)))
print("Number of features used: {}".format(np.sum(lasso00001.coef_ != 0)))
```

**Out[38]:**

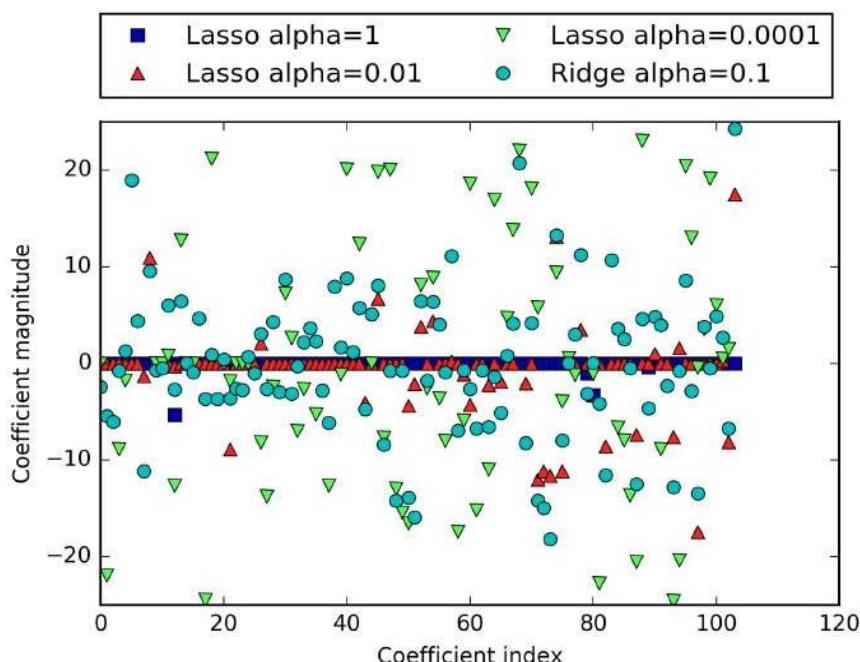
```
Training set score: 0.95
Test set score: 0.64
Number of features used: 94
```

Sekali lagi, kita dapat memplot koefisien dari model yang berbeda, mirip dengan Gambar 2-12. Hasilnya ditunjukkan pada Gambar 2.14:

**In[39]:**

```
plt.plot(lasso.coef_, 's', label="Lasso alpha=1")
plt.plot(lasso001.coef_, '^', label="Lasso alpha=0.01")
plt.plot(lasso00001.coef_, 'v', label="Lasso alpha=0.0001")

plt.plot(ridge01.coef_, 'o', label="Ridge alpha=0.1")
plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
```



**Gambar 2.14** Membandingkan besaran koefisien untuk regresi laso dengan nilai regresi alpha dan ridge yang berbeda

Untuk alpha=1, kita tidak hanya melihat bahwa sebagian besar koefisien adalah nol (yang telah kita ketahui), tetapi koefisien yang tersisa juga kecil besarnya. Menurunkan alfa menjadi 0,01, kami memperoleh solusi yang ditunjukkan sebagai titik-titik hijau, yang menyebabkan sebagian besar fitur menjadi persis nol. Menggunakan alpha=0,00001, kita mendapatkan model yang cukup tidak teratur, dengan sebagian besar koefisien bukan nol dan besarnya besar. Sebagai perbandingan, solusi Ridge terbaik ditampilkan dalam warna teal. Model Ridge dengan alpha=0.1 memiliki kinerja prediksi yang sama dengan model lasso dengan alpha=0.01, tetapi menggunakan Ridge, semua koefisien adalah bukan nol.

Dalam praktiknya, regresi ridge biasanya menjadi pilihan pertama di antara kedua model ini. Namun, jika Anda memiliki banyak fitur dan berharap hanya beberapa fitur yang penting, Lasso mungkin merupakan pilihan yang lebih baik. Demikian pula, jika Anda ingin memiliki model yang mudah diinterpretasikan, Lasso akan menyediakan model yang lebih mudah dipahami, karena hanya akan memilih subset dari fitur input. scikit-learn juga menyediakan kelas ElasticNet, yang menggabungkan hukuman Lasso dan Ridge. Dalam praktiknya, kombinasi ini bekerja paling baik, meskipun dengan harga memiliki dua parameter untuk disesuaikan: satu untuk regularisasi L1, dan satu untuk regularisasi L2.

### **Model linier untuk klasifikasi**

Model linier juga banyak digunakan untuk klasifikasi. Mari kita lihat klasifikasi biner terlebih dahulu. Dalam hal ini, prediksi dibuat menggunakan rumus berikut:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b > 0$$

Rumusnya terlihat sangat mirip dengan rumus untuk regresi linier, tetapi alih-alih hanya mengembalikan jumlah bobot fitur, kami membatasi nilai prediksi pada nol. Jika fungsi lebih kecil dari nol, kami memprediksi kelas -1; jika lebih besar dari nol, kami memprediksi kelas +1. Aturan prediksi ini umum untuk semua model linier untuk klasifikasi. Sekali lagi, ada banyak cara berbeda untuk menemukan koefisien ( $w$ ) dan intersep ( $b$ ).

Untuk model linier untuk regresi, output, , adalah fungsi linier dari fitur: garis, bidang, atau hyperplane (dalam dimensi yang lebih tinggi). Untuk model linier untuk klasifikasi, batas keputusan adalah fungsi linier dari input. Dengan kata lain, pengklasifikasi linier (biner) adalah pengklasifikasi yang memisahkan dua kelas menggunakan garis, bidang, atau hyperplane. Kita akan melihat contohnya di bagian ini.

Ada banyak algoritma untuk mempelajari model linier. Semua algoritma ini berbeda dalam dua cara berikut:

- Cara mereka mengukur seberapa baik kombinasi tertentu dari koefisien dan intersep cocok dengan data pelatihan
- Jika dan jenis regularisasi apa yang mereka gunakan

Algoritma yang berbeda memilih cara yang berbeda untuk mengukur apa yang dimaksud dengan "menyesuaikan set pelatihan dengan baik". Untuk alasan matematis teknis, tidak mungkin untuk menyesuaikan  $w$  dan  $b$  untuk meminimalkan jumlah kesalahan klasifikasi yang dihasilkan algoritma, seperti yang diharapkan. Untuk tujuan kita, dan banyak aplikasi, pilihan yang berbeda untuk item 1 dalam daftar sebelumnya (disebut fungsi kerugian) tidak begitu penting.

Dua algoritma klasifikasi linier yang paling umum adalah regresi logistik, diimplementasikan dalam `model_linier`. Regresi Logistik, dan mesin vektor pendukung linier (SVM linier), diimplementasikan dalam `svm.LinearSVC` (SVC adalah singkatan dari pengklasifikasi vektor dukungan). Terlepas dari namanya, `LogisticRegression` adalah algoritma klasifikasi dan bukan algoritma regresi, dan tidak harus bingung dengan `LinearRegression`.

Kita dapat menerapkan model `LogisticRegression` dan `LinearSVC` ke dataset tempa, dan memvisualisasikan batas keputusan seperti yang ditemukan oleh model linier (Gambar 2-15):

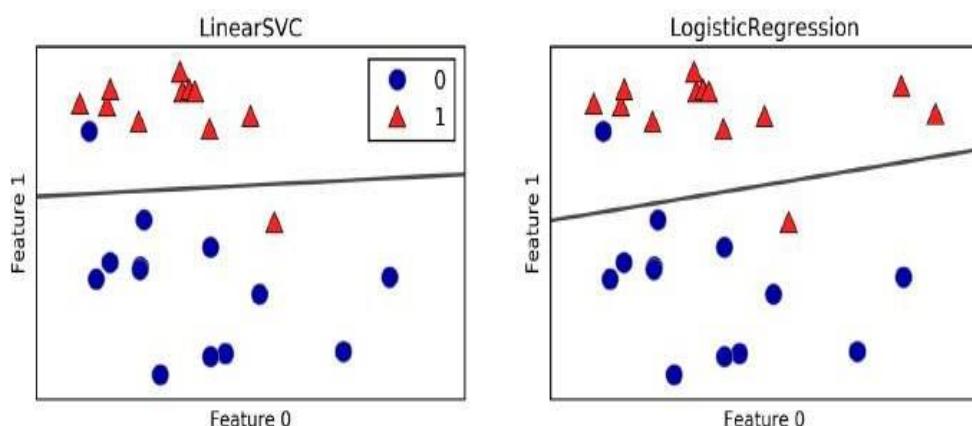
**In[40]:**

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

X, y = mglearn.datasets.make_forge()

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
                                    ax=ax, alpha=.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
axes[0].legend()
```



**Gambar 2.15** Batas keputusan dari SVM linier dan regresi logistik pada kumpulan data palsu dengan parameter default

Pada gambar ini, kita memiliki fitur pertama dari kumpulan data tempa pada sumbu x dan fitur kedua pada sumbu y, seperti sebelumnya. Kami menampilkan batas keputusan yang ditemukan oleh `LinearSVC` dan `LogisticRegression` masing-masing sebagai garis lurus, memisahkan area yang diklasifikasikan sebagai kelas 1 di bagian atas dari area yang diklasifikasikan sebagai kelas 0 di bagian bawah. Dengan kata lain, setiap titik data baru yang terletak di atas garis hitam akan diklasifikasikan sebagai kelas 1 oleh pengklasifikasi masing-

masing, sedangkan setiap titik yang terletak di bawah garis hitam akan diklasifikasikan sebagai kelas 0.

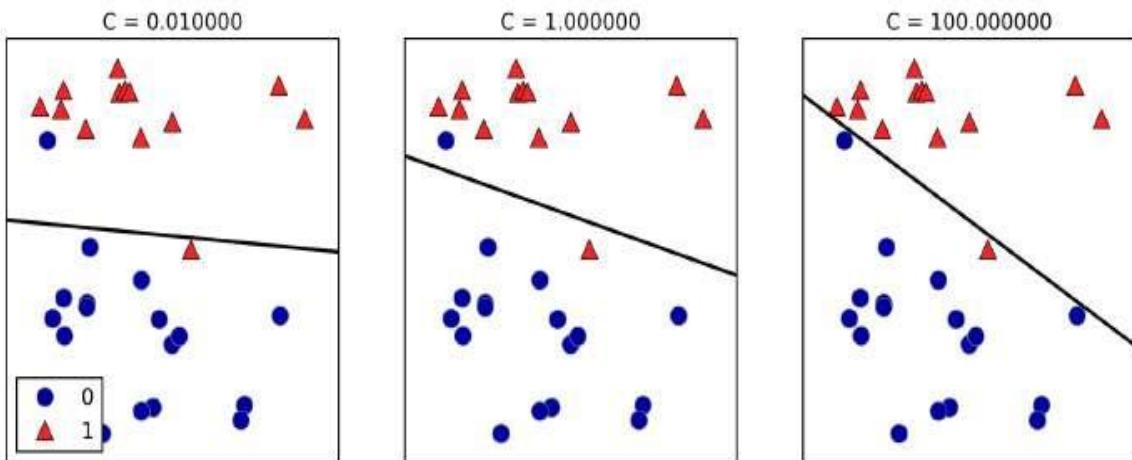
Kedua model muncul dengan batasan keputusan yang serupa. Perhatikan bahwa keduanya salah mengklasifikasikan dua poin. Secara default, kedua model menerapkan regularisasi L2, dengan cara yang sama seperti yang dilakukan Ridge untuk regresi.

Untuk LogisticRegression dan LinearSVC parameter trade-off yang menentukan kekuatan regularisasi disebut  $C$ , dan nilai  $C$  yang lebih tinggi sesuai dengan regularisasi yang lebih sedikit. Dengan kata lain, ketika Anda menggunakan nilai tinggi untuk parameter  $C$ , LogisticRegression dan LinearSVC mencoba menyesuaikan set pelatihan sebaik mungkin, sementara dengan nilai parameter  $C$  yang rendah, model lebih menekankan pada pencarian vektor koefisien ( $w$ ) yang mendekati nol.

Ada aspek lain yang menarik tentang cara kerja parameter  $C$ . Menggunakan nilai  $C$  yang rendah akan menyebabkan algoritme mencoba menyesuaikan dengan "majoritas" titik data, sementara menggunakan nilai  $C$  yang lebih tinggi menekankan pentingnya setiap titik data individual diklasifikasikan dengan benar. Berikut adalah ilustrasi menggunakan LinearSVC (Gambar 2.16):

In[41]:

```
mglearn.plots.plot_linear_svc_regularization()
```



Gambar 2.16 Batas keputusan SVM linier pada kumpulan data forge untuk nilai  $C$  . yang berbeda

Di sisi kiri, kami memiliki  $C$  yang sangat kecil yang sesuai dengan banyak regularisasi. Sebagian besar poin di kelas 0 berada di atas, dan sebagian besar poin di kelas 1 berada di bawah. Model yang sangat teratur memilih garis yang relatif horizontal, salah mengklasifikasikan dua titik. Di plot tengah,  $C$  sedikit lebih tinggi, dan model lebih berfokus pada dua sampel yang salah diklasifikasikan, memiringkan batas keputusan. Akhirnya, di sisi kanan, nilai  $C$  yang sangat tinggi dalam model banyak memiringkan batas keputusan, sekarang mengklasifikasikan semua titik di kelas 0 dengan benar. Salah satu titik di kelas 1 masih salah diklasifikasikan, karena tidak mungkin mengklasifikasikan semua titik dalam kumpulan data ini dengan benar menggunakan garis lurus. Model yang diilustrasikan di sisi kanan berusaha keras

untuk mengklasifikasikan semua poin dengan benar, tetapi mungkin tidak menangkap tata letak kelas secara keseluruhan dengan baik. Dengan kata lain, model ini cenderung overfitting.

Sama halnya dengan kasus regresi, model linier untuk klasifikasi mungkin tampak sangat membatasi dalam ruang berdimensi rendah, hanya memungkinkan untuk batas keputusan yang berupa garis lurus atau bidang. Sekali lagi, dalam dimensi tinggi, model linier untuk klasifikasi menjadi sangat kuat, dan menjaga dari overfitting menjadi semakin penting ketika mempertimbangkan lebih banyak fitur.

Mari kita analisis Linear Logistic lebih detail pada dataset Kanker Payudara:

**In[42]:**

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression().fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

**Out[42]:**

```
Training set score: 0.953
Test set score: 0.958
```

Nilai default C=1 memberikan kinerja yang cukup baik, dengan akurasi 95% pada set pelatihan dan pengujian. Tetapi karena kinerja set pelatihan dan pengujian sangat dekat, kemungkinan kami kurang fit. Mari kita coba meningkatkan C agar sesuai dengan model yang lebih fleksibel:

**In[43]:**

```
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg100.score(X_test, y_test)))
```

**Out[43]:**

```
Training set score: 0.972
Test set score: 0.965
```

Menggunakan C=100 menghasilkan akurasi set pelatihan yang lebih tinggi, dan juga akurasi set pengujian yang sedikit meningkat, mengkonfirmasikan intuisi kami bahwa model yang lebih kompleks harus berperforma lebih baik.

Kami juga dapat menyelidiki apa yang terjadi jika kami menggunakan model yang lebih teratur daripada default C=1, dengan menyetel C=0,01:

In[44]:

```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg001.score(X_test, y_test)))
```

Out[44]:

```
Training set score: 0.934
Test set score: 0.930
```

Seperti yang diharapkan, ketika bergerak lebih ke kiri sepanjang skala yang ditunjukkan pada Gambar 2-1 dari model yang sudah underfit, akurasi set pelatihan dan pengujian menurun relatif terhadap parameter default.

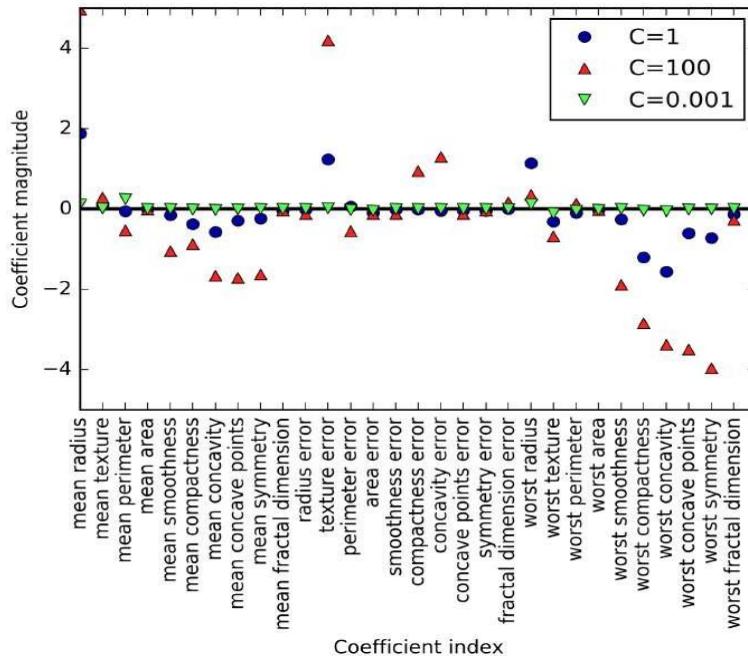
Akhirnya, mari kita lihat koefisien yang dipelajari oleh model dengan tiga pengaturan yang berbeda dari parameter regularisasi C (Gambar 2-17):

In[45]:

```
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.legend()
```



Karena LogisticRegression menerapkan regularisasi L2 secara default, hasilnya terlihat mirip dengan yang dihasilkan oleh Ridge pada Gambar 2-12. Regularisasi yang lebih kuat mendorong koefisien semakin mendekati nol, meskipun koefisien tidak pernah benar-benar nol. Memeriksa plot lebih dekat, kita juga dapat melihat efek yang menarik pada koefisien ketiga, untuk "keliling rata-rata." Untuk C=100 dan C=1, koefisiennya negatif, sedangkan untuk C=0,001, koefisiennya positif, bahkan lebih besar daripada untuk C=1. Menafsirkan model seperti ini, orang mungkin berpikir bahwa koefisien memberi tahu kita kelas mana yang terkait dengan fitur. Misalnya, orang mungkin berpikir bahwa fitur "kesalahan teks" yang tinggi terkait dengan sampel yang "ganous". Namun, perubahan tanda dalam koefisien untuk "perbatasan rata-rata" berarti bahwa tergantung pada model mana yang kita lihat, "keliling rata-rata" yang tinggi dapat dianggap sebagai indikasi "jinak" atau indikasi "ganous." Ini menggambarkan bahwa interpretasi koefisien model linier harus selalu diambil dengan sebutir garam.



**Gambar 2.17** Koefisien yang dipelajari dengan regresi logistik pada dataset Kanker Payudara untuk nilai C yang berbeda

Jika kita menginginkan model yang lebih dapat diinterpretasikan, menggunakan regularisasi L1 mungkin dapat membantu, karena model tersebut membatasi hanya menggunakan beberapa fitur. Berikut adalah plot koefisien dan akurasi klasifikasi untuk regularisasi L1 (Gambar 2.18):

In[46]:

```

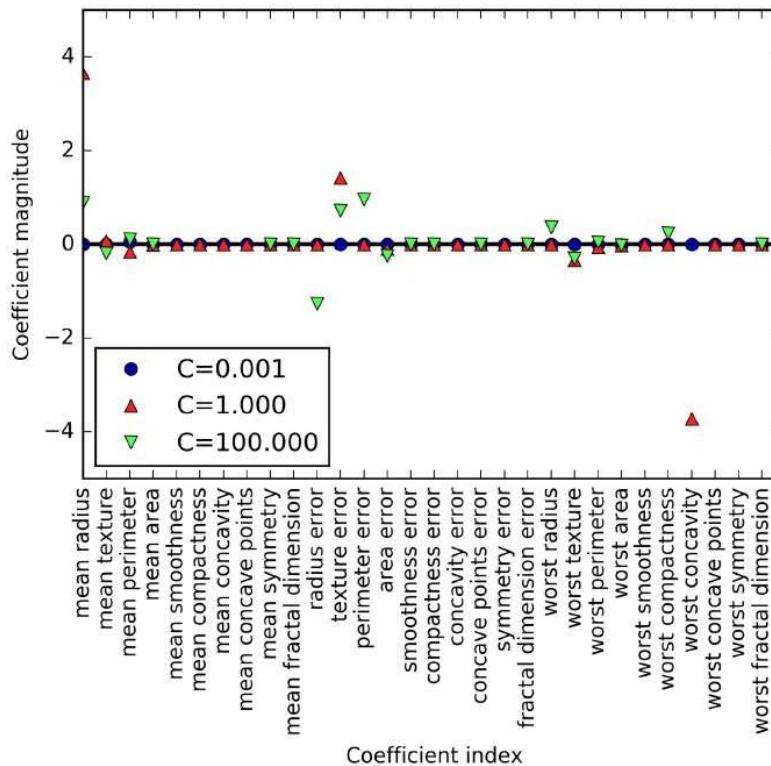
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    lr_l1 = LogisticRegression(C=C, penalty="l1").fit(X_train, y_train)
    print("Training accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_train, y_train)))
    print("Test accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_test, y_test)))
    plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))

plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")

plt.ylim(-5, 5)
plt.legend(loc=3)

```

Seperti yang Anda lihat, ada banyak persamaan antara model linier untuk klasifikasi biner dan model linier untuk regresi. Seperti dalam regresi, perbedaan utama antara model adalah parameter penalti, yang mempengaruhi regularisasi dan apakah model akan menggunakan semua fitur yang tersedia atau hanya memilih subset.



**Gambar 2-18.** Koefisien yang dipelajari dengan regresi logistik dengan penalti L1 pada dataset Kanker Payudara untuk nilai C yang berbeda

### Model linier untuk klasifikasi multikelas

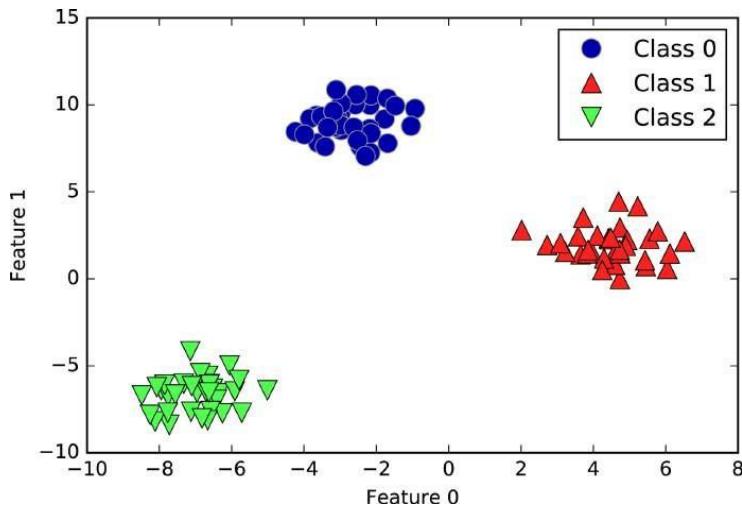
Banyak model klasifikasi linier hanya untuk klasifikasi biner, dan tidak meluas secara alami ke kasus multikelas (dengan pengecualian regresi logistik). Teknik umum untuk memperluas algoritma klasifikasi biner ke algoritma klasifikasi multikelas adalah pendekatan satu lawan satu. Dalam pendekatan satu lawan satu, model biner dipelajari untuk setiap kelas yang mencoba memisahkan kelas itu dari semua kelas lainnya, menghasilkan model biner sebanyak kelas yang ada. Untuk membuat prediksi, semua pengklasifikasi biner dijalankan pada titik uji. Pengklasifikasi yang memiliki skor tertinggi pada kelas tunggalnya “menang”, dan label kelas ini dikembalikan sebagai prediksi.

$$w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

In[47]:

```
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(["Class 0", "Class 1", "Class 2"])
```



**Gambar 2.19** Dataset mainan dua dimensi yang berisi tiga kelas

Sekarang, kita melatih classifier LinearSVC pada dataset:

**In[48]:**

```
linear_svm = LinearSVC().fit(X, y)
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
```

**Out[48]:**

```
Coefficient shape: (3, 2)
Intercept shape: (3,)
```

Kita melihat bahwa bentuk koefisien\_ adalah (3, 2), artinya setiap baris koefisien\_ berisi vektor koefisien untuk salah satu dari tiga kelas dan setiap kolom menyimpan nilai koefisien untuk fitur tertentu (ada dua di Himpunan data). Intersep\_ sekarang menjadi array satu dimensi, menyimpan intersep untuk setiap kelas. Mari kita visualisasikan garis yang diberikan oleh tiga pengklasifikasi biner (Gambar 2.20):

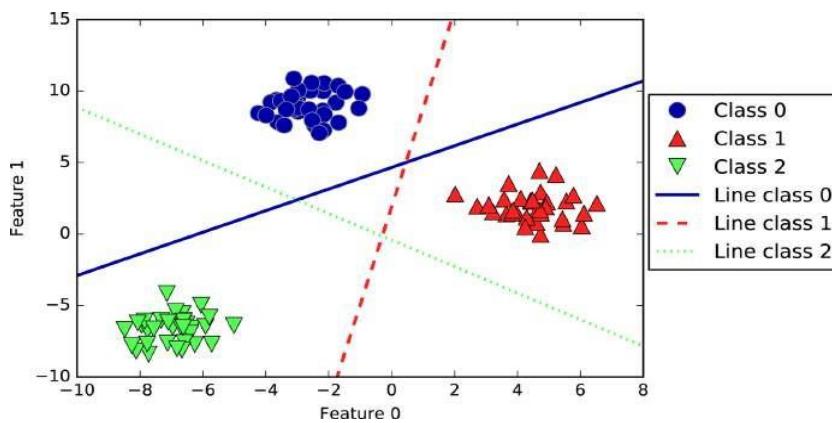
**In[49]:**

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
'Line class 2'], loc=(1.01, 0.3))
```

Anda dapat melihat bahwa semua titik yang termasuk dalam kelas 0 dalam data pelatihan berada di atas garis yang sesuai dengan kelas 0, yang berarti mereka berada di sisi "kelas 0" dari pengklasifikasi biner ini. Titik-titik di kelas 0 berada di atas garis yang sesuai dengan kelas 2, yang berarti mereka diklasifikasikan sebagai "beristirahat" oleh pengklasifikasi biner untuk kelas 2. Titik-titik milik kelas 0 berada di sebelah kiri garis yang sesuai dengan kelas

1, yang berarti pengklasifikasi biner untuk kelas 1 juga mengklasifikasikannya sebagai "istirahat." Oleh karena itu, setiap titik di area ini akan diklasifikasikan sebagai kelas 0 oleh pengklasifikasi akhir (hasil dari rumus kepercayaan klasifikasi untuk pengklasifikasi 0 lebih besar dari nol, sedangkan untuk dua kelas lainnya lebih kecil dari nol).

Tapi bagaimana dengan segitiga di tengah plot? Ketiga pengklasifikasi biner mengklasifikasikan titik di sana sebagai "istirahat." Kelas mana yang akan diberikan titik di sana? Jawabannya adalah yang memiliki nilai tertinggi untuk rumus klasifikasi: kelas dari garis terdekat.

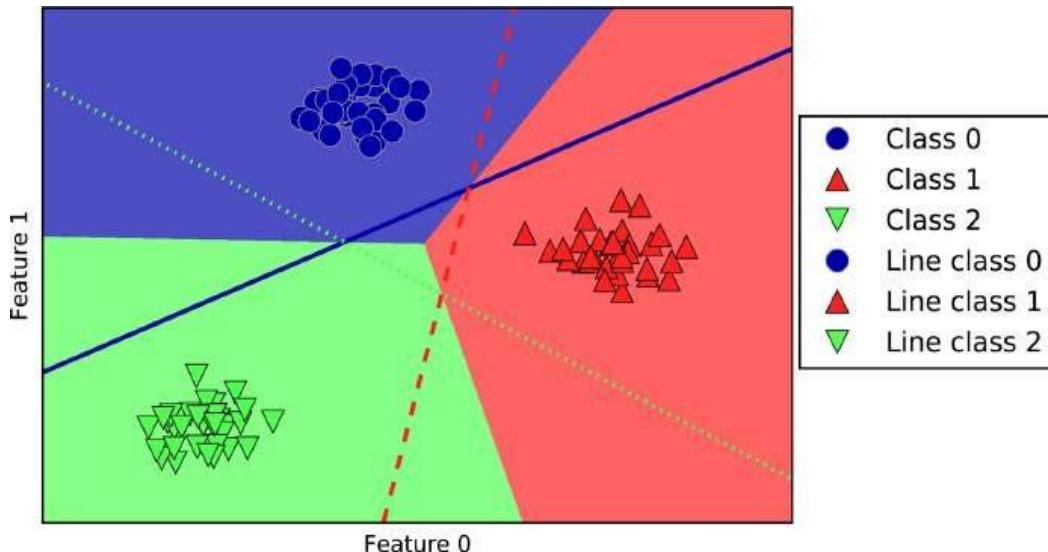


**Gambar 2.20** Batas-batas keputusan dipelajari oleh tiga pengklasifikasi satu lawan istirahat

Contoh berikut (Gambar 2.21) menunjukkan prediksi untuk semua wilayah ruang 2D:

**In[50]:**

```
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
           'Line class 2'], loc=(1.01, 0.3))
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 2.21** Batas keputusan multikelas yang diturunkan dari tiga pengklasifikasi satu lawan satu

### Kekuatan, kelemahan, dan parameter

Parameter utama dari model linier adalah parameter regularisasi, yang disebut alpha dalam model regresi dan C dalam LinearSVC dan LogisticRegression. Nilai besar untuk alfa atau nilai kecil untuk C berarti model sederhana. Khususnya untuk model regresi, penyetelan parameter ini cukup penting. Biasanya C dan alpha dicari pada skala logaritmik. Keputusan lain yang harus Anda buat adalah apakah Anda ingin menggunakan regularisasi L1 atau regularisasi L2. Jika Anda berasumsi bahwa hanya beberapa fitur Anda yang benar-benar penting, Anda harus menggunakan L1. Jika tidak, Anda harus default ke L2. L1 juga dapat berguna jika interpretabilitas model itu penting. Karena L1 hanya akan menggunakan beberapa fitur, lebih mudah untuk menjelaskan fitur mana yang penting bagi model, dan apa efek dari fitur tersebut.

Model linier sangat cepat untuk dilatih, dan juga cepat untuk diprediksi. Mereka menskalakan ke kumpulan data yang sangat besar dan bekerja dengan baik dengan data yang jarang. Jika data Anda terdiri dari ratusan ribu atau jutaan sampel, Anda mungkin ingin menyelidiki menggunakan opsi `solver='sag'` di LogisticRegression dan Ridge, yang bisa lebih cepat daripada default pada kumpulan data besar. Opsi lainnya adalah kelas SGDClassifier dan kelas SGDRegressor, yang mengimplementasikan versi model linier yang lebih skalabel yang dijelaskan di sini.

Kekuatan lain dari model linier adalah membuatnya relatif mudah untuk memahami bagaimana prediksi dibuat, menggunakan rumus yang kita lihat sebelumnya untuk regresi dan klasifikasi. Sayangnya, seringkali tidak sepenuhnya jelas mengapa koefisien seperti itu. Ini terutama benar jika kumpulan data Anda memiliki fitur yang sangat berkorelasi; dalam kasus ini, koefisien mungkin sulit untuk ditafsirkan.

Model linier sering berkinerja baik ketika jumlah fitur lebih besar dibandingkan dengan jumlah sampel. Mereka juga sering digunakan pada kumpulan data yang sangat besar, hanya karena tidak layak untuk melatih model lain. Namun, dalam ruang berdimensi lebih rendah, model lain mungkin menghasilkan kinerja generalisasi yang lebih baik. Kita akan melihat

beberapa contoh di mana model linier gagal di “Mesin Vektor Dukungan Kernel” pada halaman 92.

### Metode Rantai

Metode fit dari semua model scikit-learn mengembalikan self. Ini memungkinkan Anda untuk menulis kode seperti berikut, yang telah kita gunakan secara ekstensif dalam bab ini:

In[51]:

```
# instantiate model and fit it in one line
logreg = LogisticRegression().fit(X_train, y_train)
```

Di sini, kami menggunakan nilai pengembalian fit (yaitu self) untuk menetapkan model yang dilatih ke variabel logreg. Rangkaian pemanggilan metode ini (di sini init dan kemudian cocok) dikenal sebagai metode chaining. Aplikasi umum lain dari metode chaining di scikit-learn adalah menyesuaikan dan memprediksi dalam satu baris:

In[52]:

```
logreg = LogisticRegression()
y_pred = logreg.fit(X_train, y_train).predict(X_test)
```

Terakhir, Anda bahkan dapat melakukan instantiasi, pemasangan, dan prediksi model dalam satu baris:

In[53]:

```
y_pred = LogisticRegression().fit(X_train, y_train).predict(X_test)
```

Varian yang sangat pendek ini tidak ideal. Banyak yang terjadi dalam satu baris, yang mungkin membuat kode sulit dibaca. Selain itu, model regresi logistik yang dipasang tidak disimpan dalam variabel apa pun, jadi kami tidak dapat memeriksanya atau menggunakannya untuk memprediksi data lain.

## 2.8 PENGKLASIFIKASI NAIVE BAYES

Pengklasifikasi Naive Bayes adalah keluarga pengklasifikasi yang sangat mirip dengan model linier yang dibahas di bagian sebelumnya. Namun, mereka cenderung lebih cepat dalam latihan. Harga yang harus dibayar untuk efisiensi ini adalah bahwa model naive Bayes sering memberikan kinerja generalisasi yang sedikit lebih buruk daripada pengklasifikasi linier seperti LogisticRegression dan LinearSVC.

Alasan mengapa model naive Bayes sangat efisien adalah karena model tersebut mempelajari parameter dengan melihat setiap fitur satu per satu dan mengumpulkan statistik per kelas sederhana dari setiap fitur. Ada tiga jenis pengklasifikasi naive Bayes yang diimplementasikan dalam scikit-learn: GaussianNB, BernoulliNB, dan MultinomialNB. GaussianNB dapat diterapkan pada data kontinu apa pun, sementara BernoulliNB mengasumsikan data biner dan MultinomialNB mengasumsikan data hitungan (yaitu, bahwa setiap fitur mewakili jumlah bilangan bulat dari sesuatu, seperti seberapa sering sebuah kata muncul dalam sebuah kalimat). BernoulliNB dan MultinomialNB banyak digunakan dalam klasifikasi data teks.

Pengklasifikasi BernoulliNB menghitung seberapa sering setiap fitur dari setiap kelas tidak nol. Ini paling mudah dipahami dengan sebuah contoh:

**In[54]:**

```
X = np.array([[0, 1, 0, 1],
              [1, 0, 1, 1],
              [0, 0, 0, 1],
              [1, 0, 1, 0]])
y = np.array([0, 1, 0, 1])
```

Di sini, kami memiliki empat titik data, dengan masing-masing empat fitur biner. Ada dua kelas, 0 dan 1. Untuk kelas 0 (titik data pertama dan ketiga), fitur pertama adalah nol dua kali dan bukan nol nol kali, fitur kedua adalah nol satu kali dan bukan nol satu kali, dan seterusnya. Hitungan yang sama ini kemudian dihitung untuk titik data di kelas kedua. Menghitung entri bukan nol per kelas pada dasarnya terlihat seperti ini:

**In[55]:**

```
counts = []
for label in np.unique(y):
    # iterate over each class
    # count (sum) entries of 1 per feature
    counts[label] = X[y == label].sum(axis=0)
print("Feature counts:\n{}".format(counts))
```

**Out[55]:**

```
Feature counts:
{0: array([0, 1, 0, 2]), 1: array([2, 0, 2, 1])}
```

Dua model naive Bayes lainnya, MultinomialNB dan GaussianNB, sedikit berbeda dalam jenis statistik yang mereka hitung. MultinomialNB memperhitungkan nilai rata-rata setiap fitur untuk setiap kelas, sedangkan GaussianNB menyimpan nilai rata-rata serta standar deviasi setiap fitur untuk setiap kelas.

Untuk membuat prediksi, titik data dibandingkan dengan statistik untuk masing-masing kelas, dan kelas yang paling cocok diprediksi. Menariknya, untuk MultinomialNB dan BernoulliNB, ini mengarah pada formula prediksi yang bentuknya sama seperti pada model linier (lihat "Model linier untuk klasifikasi" di halaman 56). Sayangnya, `coef_` untuk model naive Bayes memiliki arti yang agak berbeda dari pada model linier, dalam `coef_` itu tidak sama dengan `w`.

### Kekuatan, kelemahan, dan parameter

MultinomialNB dan BernoulliNB memiliki parameter tunggal, `alfa`, yang mengontrol kompleksitas model. Cara kerja alpha adalah bahwa algoritme menambahkan ke data alfa banyak titik data virtual yang memiliki nilai positif untuk semua fitur. Ini menghasilkan "pemulusan" statistik. Alfa besar berarti lebih banyak pemulusan, menghasilkan model yang tidak terlalu rumit. Kinerja algoritme relatif kuat untuk pengaturan `alfa`, artinya pengaturan `alfa` tidak penting untuk kinerja yang baik. Namun, penyetelan biasanya meningkatkan akurasi.

GaussianNB sebagian besar digunakan pada data berdimensi sangat tinggi, sedangkan dua varian naive Bayes lainnya banyak digunakan untuk data jumlah jarang seperti teks. MultinomialNB biasanya berkinerja lebih baik daripada BinaryNB, terutama pada kumpulan data dengan jumlah fitur bukan nol yang relatif besar (yaitu, dokumen besar).

Model naive Bayes berbagi banyak kekuatan dan kelemahan dari model linier. Mereka sangat cepat untuk dilatih dan diprediksi, dan prosedur pelatihannya mudah dimengerti. Model bekerja sangat baik dengan data sparse berdimensi tinggi dan relatif kuat terhadap parameter. Model Naive Bayes adalah model dasar yang bagus dan sering digunakan pada kumpulan data yang sangat besar, di mana pelatihan bahkan model linier mungkin memakan waktu terlalu lama.

## 2.9 POHON KEPUTUSAN

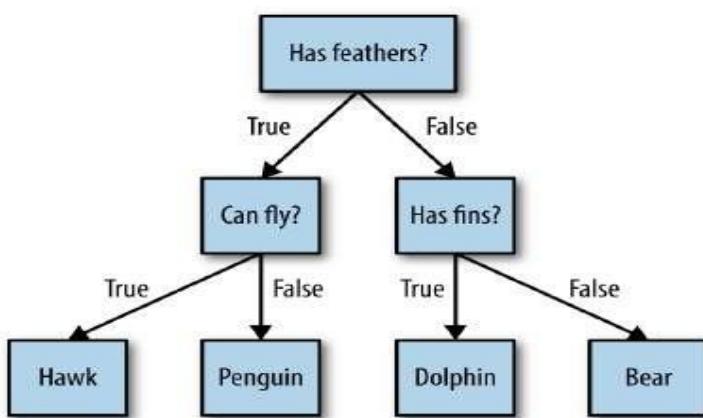
Pohon keputusan adalah model yang banyak digunakan untuk tugas klasifikasi dan regresi. Pada dasarnya, mereka mempelajari hierarki pertanyaan if/else, yang mengarah pada keputusan.

Pertanyaan-pertanyaan ini mirip dengan pertanyaan yang mungkin Anda ajukan dalam permainan 20 Pertanyaan. Bayangkan Anda ingin membedakan empat hewan berikut: beruang, elang, penguin, dan lumba-lumba. Tujuan Anda adalah mendapatkan jawaban yang benar dengan mengajukan pertanyaan if/else sesedikit mungkin. Anda dapat memulai dengan menanyakan apakah hewan tersebut memiliki bulu, sebuah pertanyaan yang mempersempit kemungkinan hewan Anda menjadi hanya dua. Jika jawabannya "ya", Anda dapat mengajukan pertanyaan lain yang dapat membantu Anda membedakan elang dan penguin. Misalnya, Anda bisa bertanya apakah hewan itu bisa terbang. Jika hewan tersebut tidak memiliki bulu, pilihan hewan yang mungkin Anda pilih adalah lumba-lumba dan beruang, dan Anda perlu mengajukan pertanyaan untuk membedakan kedua hewan ini—misalnya, menanyakan apakah hewan tersebut memiliki sirip.

Serangkaian pertanyaan ini dapat dinyatakan sebagai pohon keputusan, seperti yang ditunjukkan pada Gambar 2.22.

In[56]:

```
mglearn.plots.plot_animal_tree()
```



**Gambar 2.22** Pohon keputusan untuk membedakan beberapa hewan

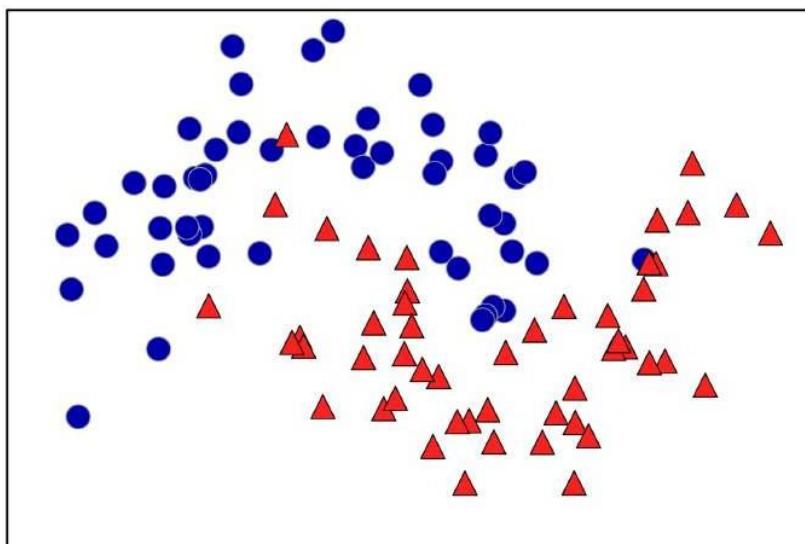
Dalam ilustrasi ini, setiap simpul di pohon mewakili pertanyaan atau simpul terminal (juga disebut daun) yang berisi jawabannya. Ujung-ujungnya menghubungkan jawaban atas pertanyaan dengan pertanyaan berikutnya yang akan Anda ajukan.

Dalam bahasa pembelajaran mesin, kami membuat model untuk membedakan antara empat kelas hewan (elang, penguin, lumba-lumba, dan beruang) menggunakan tiga fitur "memiliki bulu", "dapat terbang", dan "memiliki sirip". Alih-alih membangun model ini dengan tangan, kita dapat mempelajarinya dari data menggunakan pembelajaran terawasi.

### Membangun pohon keputusan

Mari kita melalui proses membangun pohon keputusan untuk kumpulan data klasifikasi 2D yang ditunjukkan pada Gambar 2.23. Dataset terdiri dari dua bentuk setengah bulan, dengan masing-masing kelas terdiri dari 75 titik data. Kami akan merujuk ke dataset ini sebagai two\_moons.

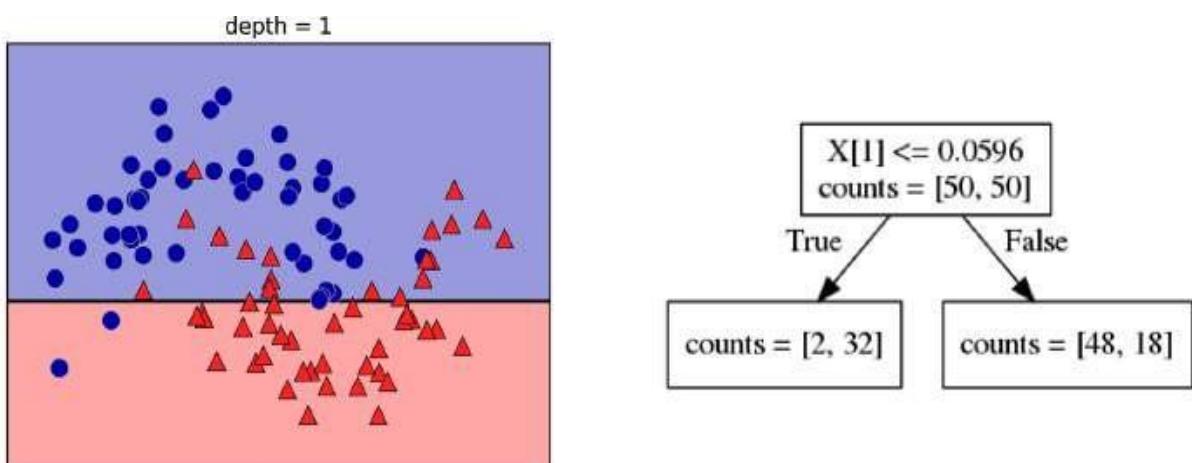
Mempelajari pohon keputusan berarti mempelajari urutan pertanyaan if/else yang membawa kita ke jawaban yang benar paling cepat. Dalam pengaturan pembelajaran mesin, pertanyaan-pertanyaan ini disebut tes (jangan dikelirukan dengan kumpulan tes, yang merupakan data yang kami gunakan untuk menguji untuk melihat seberapa dapat digeneralisasikan model kami). Biasanya data tidak datang dalam bentuk fitur biner ya/tidak seperti pada contoh hewan, melainkan direpresentasikan sebagai fitur kontinu seperti dalam kumpulan data 2D yang ditunjukkan pada Gambar 2.23. Pengujian yang digunakan pada data kontinyu berupa "Apakah fitur i lebih besar dari nilai a?"



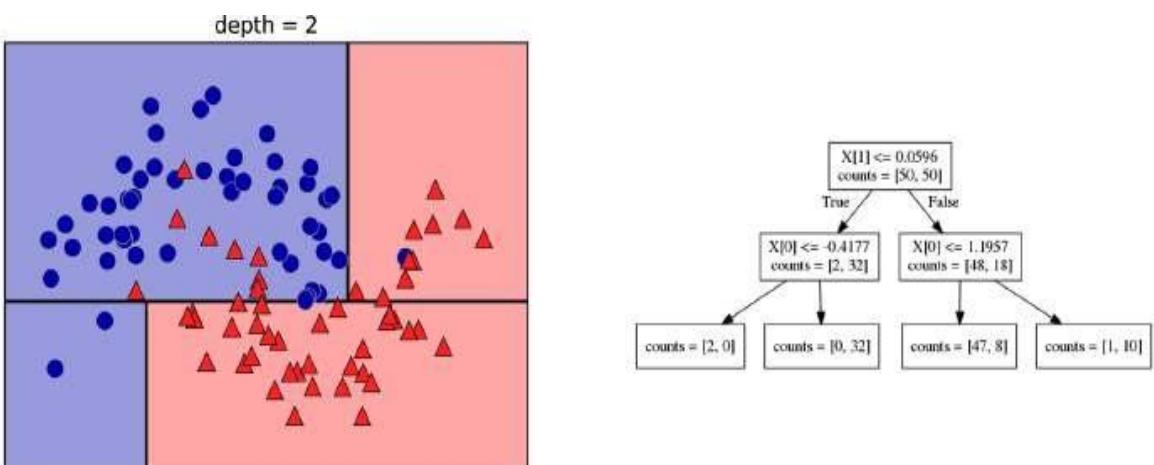
**Gambar 2.23** Dataset dua bulan di mana pohon keputusan akan dibangun

Untuk membangun pohon, algoritma mencari semua pengujian yang mungkin dan menemukan yang paling informatif tentang variabel target. Gambar 2.24 menunjukkan tes pertama yang diambil. Memisahkan dataset secara vertikal pada  $x[1] = 0,0596$  menghasilkan informasi paling banyak; itu paling baik memisahkan poin di kelas 1 dari poin di kelas 2. Node teratas, juga disebut root, mewakili seluruh dataset, terdiri dari 75 poin milik kelas 0 dan 75 poin milik kelas 1. Pemisahan dilakukan dengan menguji apakah  $x[1] \leq 0,0596$ , ditunjukkan dengan garis hitam. Jika tesnya benar, sebuah titik diberikan ke simpul kiri, yang berisi 2 poin

milik kelas 0 dan 32 poin milik kelas 1. Jika tidak, titik itu diberikan ke simpul kanan, yang berisi 48 poin milik kelas 0 dan 18 poin milik kelas 1. Kedua node ini sesuai dengan wilayah atas dan bawah yang ditunjukkan pada Gambar 2.24. Meskipun pembagian pertama melakukan pekerjaan yang baik untuk memisahkan dua kelas, wilayah bawah masih berisi poin milik kelas 0, dan wilayah atas masih berisi poin milik kelas 1. Kita dapat membangun model yang lebih akurat dengan mengulangi proses pencarian tes terbaik di kedua wilayah. Gambar 2.25 menunjukkan bahwa pembagian berikutnya yang paling informatif untuk wilayah kiri dan kanan didasarkan pada  $x[0]$ .



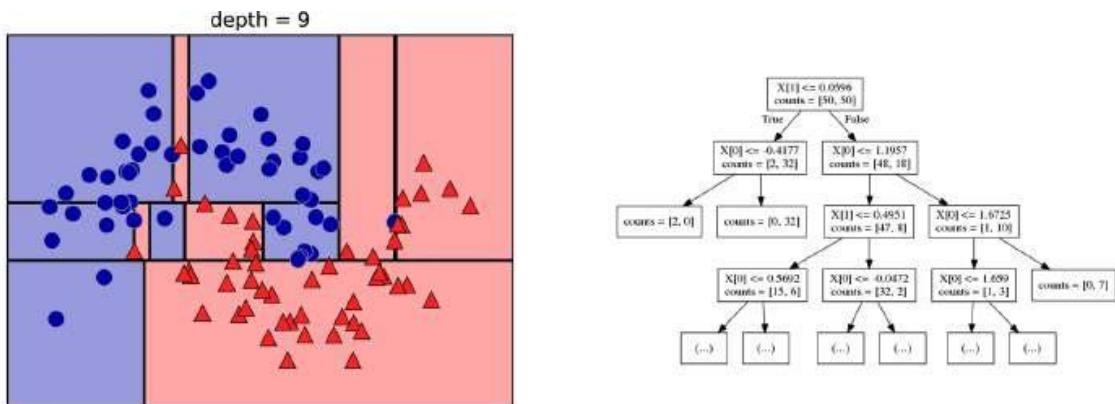
**Gambar 2.24** Batas keputusan pohon dengan kedalaman 1 (kiri) dan pohon yang sesuai (kanan)



**Gambar 2.25** Batas keputusan pohon dengan kedalaman 2 (kiri) dan pohon keputusan yang sesuai (kanan)

Proses rekursif ini menghasilkan pohon keputusan biner, dengan setiap node berisi tes. Atau, Anda dapat menganggap setiap pengujian sebagai pemisahan bagian data yang saat ini sedang dipertimbangkan di sepanjang satu sumbu. Ini menghasilkan pandangan algoritma sebagai membangun partisi hirarkis. Karena setiap pengujian hanya menyangkut satu fitur, wilayah di partisi yang dihasilkan selalu memiliki batas sumbu-paralel.

Partisi rekursif dari data diulang sampai setiap wilayah di partisi (setiap daun di pohon keputusan) hanya berisi nilai target tunggal (kelas tunggal atau nilai regresi tunggal). Daun pohon yang berisi titik data yang semuanya memiliki nilai target yang sama disebut murni. Partisi terakhir untuk dataset ini ditunjukkan pada Gambar 2.26.



**Gambar 2.26** Penetapan batas pohon dengan kedalaman 9 (kiri) dan bagian dari pohon yang sesuai (kanan); pohon penuhnya cukup besar dan sulit untuk divisualisasikan

Prediksi pada titik data baru dibuat dengan memeriksa wilayah mana dari partisi ruang fitur tempat titik tersebut berada, dan kemudian memprediksi target mayoritas (atau target tunggal dalam kasus daun murni) di wilayah tersebut. Daerah dapat ditemukan dengan melintasi pohon dari akar dan ke kiri atau kanan, tergantung pada apakah tes terpenuhi atau tidak.

Dimungkinkan juga untuk menggunakan pohon untuk tugas regresi, menggunakan teknik yang persis sama. Untuk membuat prediksi, kami melintasi pohon berdasarkan pengujian di setiap node dan menemukan daun tempat titik data baru jatuh. Output untuk titik data ini adalah target rata-rata dari titik pelatihan di daun ini.

#### Mengontrol kompleksitas pohon keputusan

Biasanya, membangun pohon seperti yang dijelaskan di sini dan berlanjut hingga semua daun murni mengarah ke model yang sangat kompleks dan sangat sesuai dengan data pelatihan. Kehadiran daun murni berarti bahwa pohon itu 100% akurat pada set pelatihan; setiap titik data dalam set pelatihan berada dalam daun yang memiliki kelas mayoritas yang benar. Overfitting dapat dilihat di sebelah kiri Gambar 2.26. Anda dapat melihat daerah-daerah yang ditentukan milik kelas 1 di tengah semua titik milik kelas 0. Di sisi lain, ada garis kecil yang diprediksi sebagai kelas 0 di sekitar titik milik kelas 0 di paling kanan. Ini bukan bagaimana orang akan membayangkan batas keputusan untuk dilihat, dan batas keputusan banyak berfokus pada titik outlier tunggal yang jauh dari titik lain di kelas itu.

Ada dua strategi umum untuk mencegah overfitting: menghentikan pembuatan pohon lebih awal (juga disebut pra-pemangkasan), atau membangun pohon tetapi kemudian menghapus atau mencutkan simpul yang berisi sedikit informasi (juga disebut pasca-pemangkasan atau hanya pemangkasan). Kriteria yang mungkin untuk pra-pemangkasan termasuk membatasi kedalaman maksimum pohon, membatasi jumlah maksimum daun, atau membutuhkan jumlah titik minimum dalam sebuah simpul untuk terus membelahnya.

Pohon keputusan dalam scikit-learn diimplementasikan dalam kelas DecisionTreeRegressor dan DecisionTreeClassifier. scikit-learn hanya mengimplementasikan pra-pangkasan, bukan pasca-pemangkasan.

Mari kita lihat efek pra-pangkasan lebih detail pada dataset Kanker Payudara. Seperti biasa, kami mengimpor kumpulan data dan membaginya menjadi bagian pelatihan dan pengujian. Kemudian kita bangun model menggunakan pengaturan default untuk mengembangkan pohon sepenuhnya (menumbuhkan pohon hingga semua daun murni). Kami memperbaiki random\_state di pohon, yang digunakan untuk tiebreak secara internal:

**In[58]:**

```
from sklearn.tree import DecisionTreeClassifier

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

**Out[58]:**

```
Accuracy on training set: 1.000
Accuracy on test set: 0.937
```

Seperti yang diharapkan, akurasi pada set pelatihan adalah 100%—karena daunnya murni, pohnnya tumbuh cukup dalam sehingga dapat dengan sempurna mengingat semua label pada data pelatihan. Akurasi set tes sedikit lebih buruk daripada model linier yang kami lihat sebelumnya, yang memiliki akurasi sekitar 95%.

Jika kita tidak membatasi kedalaman pohon keputusan, pohon dapat menjadi sewenang-wenang dalam dan kompleks. Oleh karena itu, pohon yang tidak dipangkas rentan terhadap overfitting dan tidak dapat digeneralisasi dengan baik ke data baru. Sekarang mari kita terapkan pra-pemangkasan ke pohon, yang akan menghentikan pengembangan pohon sebelum kita benar-benar cocok dengan data pelatihan. Salah satu pilihannya adalah berhenti membangun pohon setelah mencapai kedalaman tertentu. Di sini kita menetapkan max\_depth=4, artinya hanya empat pertanyaan berurutan yang dapat diajukan (lih. Gambar 2.24 dan 2.26). Membatasi kedalaman pohon mengurangi overfitting. Hal ini menyebabkan akurasi yang lebih rendah pada set pelatihan, tetapi peningkatan pada set pengujian:

**In[59]:**

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

**Out[59]:**

```
Accuracy on training set: 0.988
Accuracy on test set: 0.951
```

## Menganalisis pohon keputusan

Kita dapat memvisualisasikan pohon menggunakan fungsi `export_graphviz` dari modul pohon. Ini menulis file dalam format file `.dot`, yang merupakan format file teks untuk menyimpan grafik. Kami menetapkan opsi untuk mewarnai node untuk mencerminkan kelas mayoritas di setiap node dan meneruskan nama kelas dan fitur sehingga pohon dapat diberi label dengan benar:

In[61]:

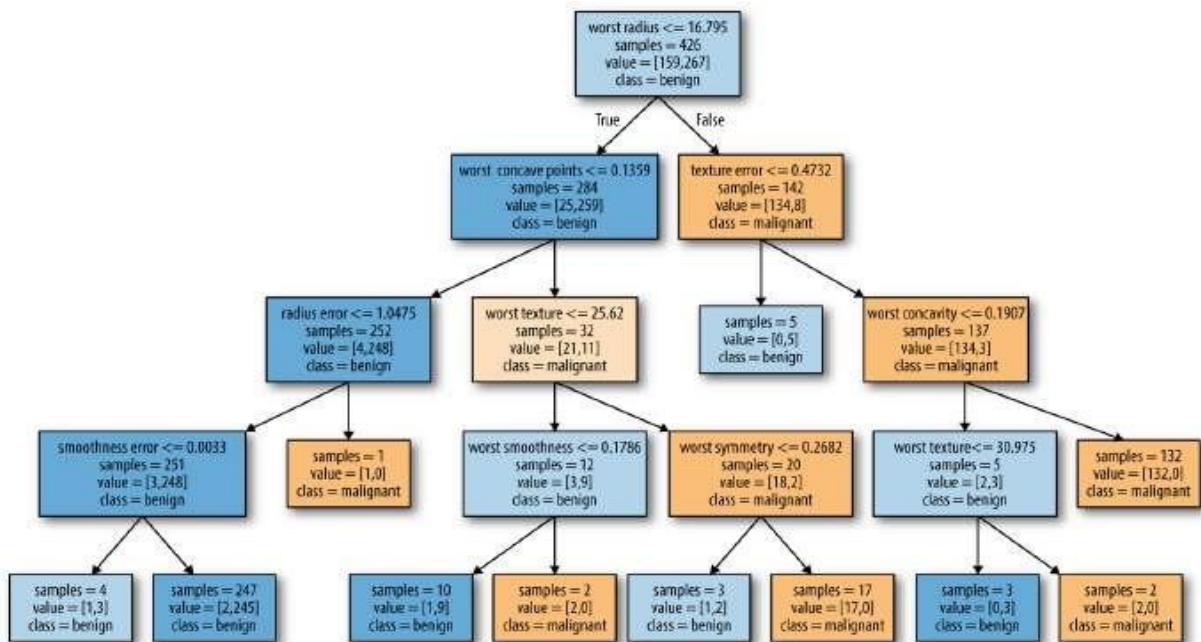
```
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"],
                feature_names=cancer.feature_names, impurity=False, filled=True)
```

Kita dapat membaca file ini dan memvisualisasikannya, seperti terlihat pada Gambar 2.27, menggunakan modul graphviz (atau Anda dapat menggunakan program apa pun yang dapat membaca file `.dot`):

In[61]:

```
import graphviz

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



Gambar 2.27 Visualisasi pohon keputusan yang dibangun di atas dataset Kanker Payudara

Visualisasi pohon memberikan pandangan mendalam yang bagus tentang bagaimana algoritme membuat prediksi, dan merupakan contoh bagus dari algoritme pembelajaran mesin yang mudah dijelaskan kepada yang tidak ahli. Namun, bahkan dengan pohon dengan kedalaman empat, seperti yang terlihat di sini, pohon itu bisa menjadi sedikit berlebihan. Pohon yang lebih dalam (kedalaman 10 tidak jarang) bahkan lebih sulit untuk dipahami. Salah satu metode untuk memeriksa pohon yang mungkin berguna adalah untuk mengetahui jalur mana yang sebenarnya diambil sebagian besar data. `N_samples` yang ditunjukkan pada setiap

node pada Gambar 2.27 memberikan jumlah sampel dalam node tersebut, sedangkan nilai memberikan jumlah sampel per kelas. Mengikuti cabang ke kanan, kita melihat bahwa radius terburuk  $\leq 16.795$  membuat simpul yang hanya berisi 8 sampel jinak tetapi 134 sampel ganas. Sisa dari sisi pohon ini kemudian menggunakan beberapa perbedaan yang lebih halus untuk memisahkan 8 sampel jinak yang tersisa ini. Dari 142 sampel yang mengarah ke kanan pada pemisahan awal, hampir semuanya (132) berakhir di daun paling kanan.

Mengambil kiri di root, untuk radius terburuk  $> 16.795$  kita berakhir dengan 25 ganas dan 259 sampel jinak. Hampir semua sampel jinak berakhir di daun kedua dari kanan, dengan sebagian besar daun lainnya mengandung sangat sedikit sampel.

### **Fitur penting di pohon**

Alih-alih melihat keseluruhan pohon, yang dapat membebani, ada beberapa properti berguna yang dapat kita peroleh untuk meringkas cara kerja pohon. Ringkasan yang paling umum digunakan adalah fitur penting, yang menilai seberapa penting setiap fitur untuk keputusan yang dibuat pohon. Ini adalah angka antara 0 dan 1 untuk setiap fitur, di mana 0 berarti "tidak digunakan sama sekali" dan 1 berarti "memprediksi target dengan sempurna." Kepentingan fitur selalu berjumlah 1:

**In[62]:**

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

**Out[62]:**

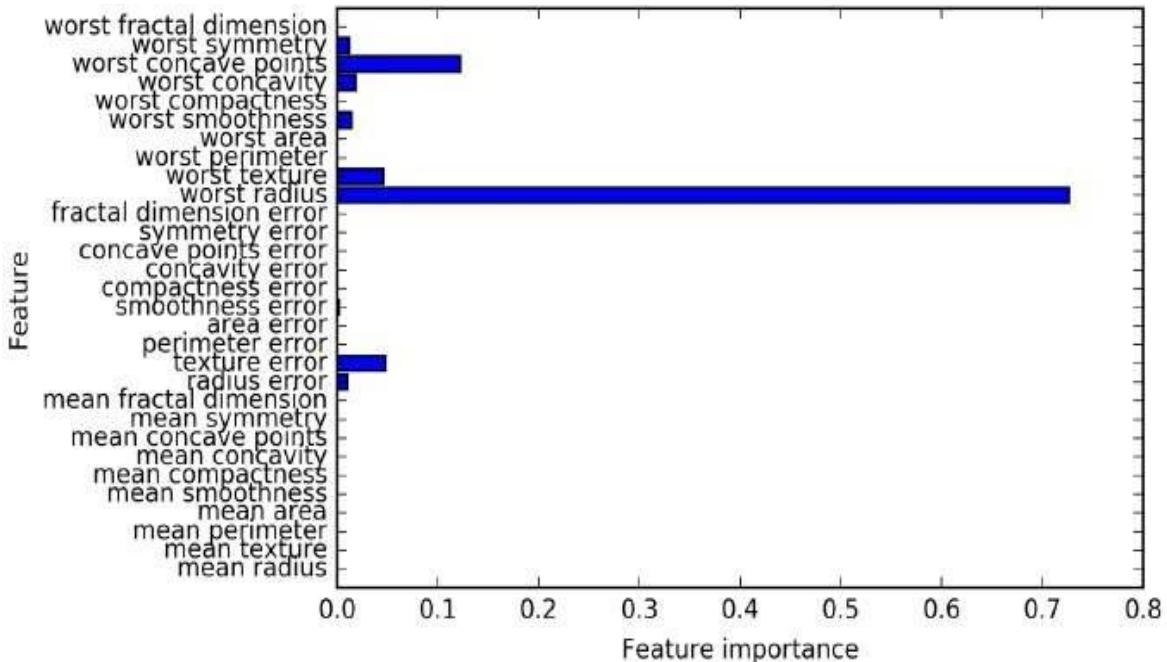
```
Feature importances:
[ 0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.         0.01
  0.048     0.         0.         0.002     0.         0.         0.         0.         0.         0.         0.727     0.046
  0.         0.         0.014     0.         0.018     0.122     0.012     0.         ]
```

Kita dapat memvisualisasikan pentingnya fitur dengan cara yang mirip dengan cara kita memvisualisasikan koefisien dalam model linier (Gambar 2.28):

**In[63]:**

```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")

plot_feature_importances_cancer(tree)
```



**Gambar 2.28** Kepentingan fitur dihitung dari pohon keputusan yang dipelajari pada dataset Kanker Payudara

Di sini kita melihat bahwa fitur yang digunakan di bagian atas ("radius terburuk") sejauh ini merupakan fitur yang paling penting. Hal ini menegaskan pengamatan kami dalam menganalisis pohon bahwa tingkat pertama sudah memisahkan dua kelas dengan cukup baik.

Namun, jika suatu fitur memiliki `feature_importance` yang rendah, bukan berarti fitur tersebut tidak informatif. Ini hanya berarti bahwa fitur tersebut tidak dipilih oleh pohon, kemungkinan karena fitur lain mengkodekan informasi yang sama.

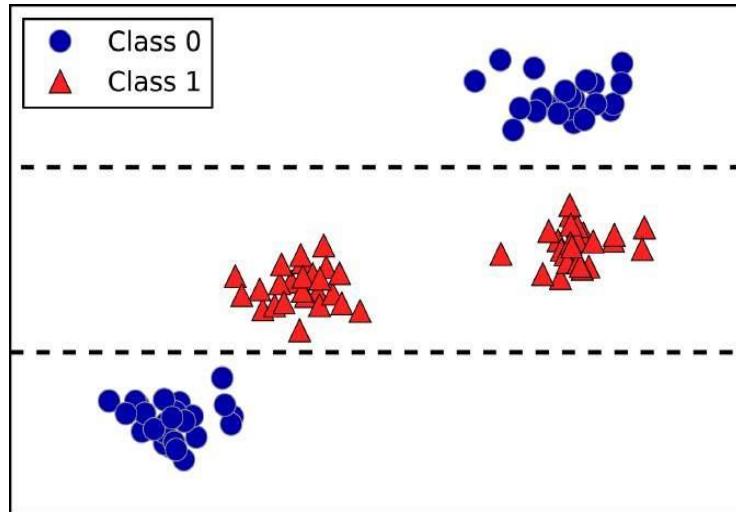
Berbeda dengan koefisien dalam model linier, kepentingan fitur selalu positif, dan tidak mengkodekan kelas mana yang merupakan indikasi fitur. Kepentingan fitur memberi tahu kita bahwa "radius terburuk" itu penting, tetapi bukan apakah radius tinggi menunjukkan sampel jinak atau ganas. Faktanya, mungkin tidak ada hubungan yang sederhana antara fitur dan kelas, seperti yang Anda lihat pada contoh berikut (Gambar 2.29 dan 2.30):

**In[64]:**

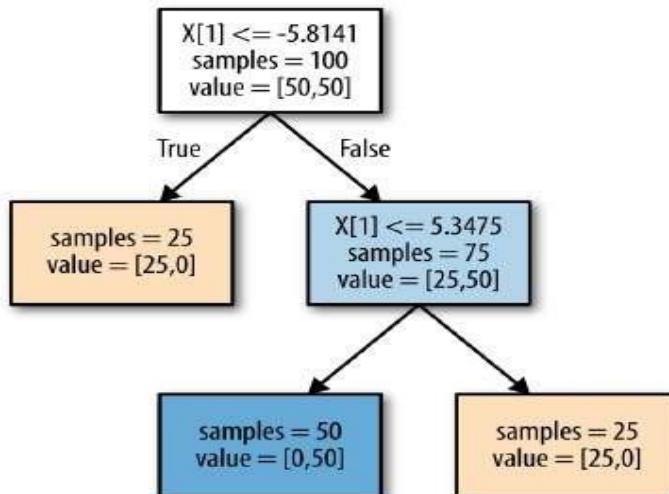
```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```

**Out[64]:**

```
Feature importances: [ 0.  1.]
```



**Gambar 2.29** Dataset dua dimensi di mana fitur pada sumbu y memiliki hubungan yang tidak monoton dengan label kelas, dan batas keputusan yang ditemukan oleh pohon keputusan



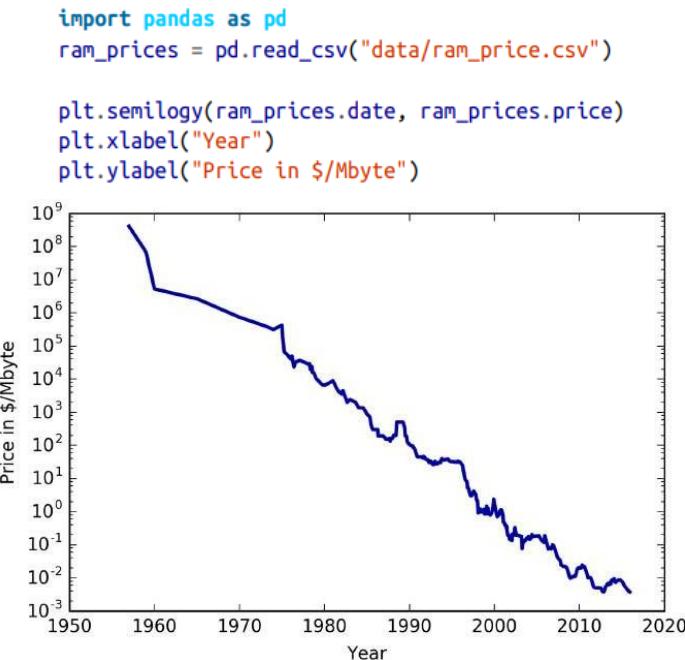
**Gambar 2.30** Pohon keputusan dipelajari pada data yang ditunjukkan pada Gambar 2.29

Plot menunjukkan dataset dengan dua fitur dan dua kelas. Di sini, semua informasi terkandung dalam  $X[1]$ , dan  $X[0]$  tidak digunakan sama sekali. Namun hubungan antara  $X[1]$  dan kelas keluaran tidak monoton, artinya kita tidak dapat mengatakan “nilai  $X[0]$  yang tinggi berarti kelas 0, dan nilai yang rendah berarti kelas 1” (atau sebaliknya).

Sementara kami memfokuskan diskusi kami di sini pada pohon keputusan untuk klasifikasi, semua yang dikatakan juga berlaku untuk pohon keputusan untuk regresi, seperti yang diimplementasikan dalam `DecisionTreeRegressor`. Penggunaan dan analisis pohon regresi sangat mirip dengan pohon klasifikasi. Namun, ada satu properti khusus dalam menggunakan model berbasis pohon untuk regresi yang ingin kami tunjukkan. `DecisionTreeRegressor` (dan semua model regresi berbasis pohon lainnya) tidak dapat melakukan ekstrapolasi, atau membuat prediksi di luar rentang data pelatihan.

Mari kita lihat ini lebih detail, menggunakan dataset harga memori komputer (RAM) historis. Gambar 2.31 menunjukkan dataset, dengan tanggal pada sumbu x dan harga satu megabyte RAM pada tahun tersebut pada sumbu y:

In[65]:



**Gambar 2.31** Perkembangan historis harga RAM, diplot pada skala log

Perhatikan skala logaritmik dari sumbu y. Ketika memplot secara logaritmik, hubungan tersebut tampaknya cukup linier dan seharusnya relatif mudah untuk diprediksi, terlepas dari beberapa gundukan.

Kami akan membuat perkiraan untuk tahun-tahun setelah tahun 2000 menggunakan data historis hingga saat itu, dengan tanggal sebagai satu-satunya fitur kami. Kami akan membandingkan dua model sederhana: DecisionTreeRegressor dan LinearRegression. Kami mengubah skala harga menggunakan logaritma, sehingga hubungannya relatif linier. Ini tidak membuat perbedaan untuk DecisionTreeRegressor, tetapi membuat perbedaan besar untuk LinearRegression (kita akan membahas ini lebih mendalam di Bab 4). Setelah melatih model dan membuat prediksi, kami menerapkan peta eksponensial untuk membatalkan transformasi logaritma. Kami membuat prediksi pada seluruh kumpulan data untuk tujuan visualisasi di sini, tetapi untuk evaluasi kuantitatif kami hanya akan mempertimbangkan kumpulan data uji:

In[66]:

```
from sklearn.tree import DecisionTreeRegressor
# use historical data to forecast prices after the year 2000
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# predict prices based on date
X_train = data_train.date[:, np.newaxis]
# we use a log-transform to get a simpler relationship of data to target
y_train = np.log(data_train.price)
```

```

tree = DecisionTreeRegressor().fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)

# predict on all data
X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# undo log-transform
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)

```

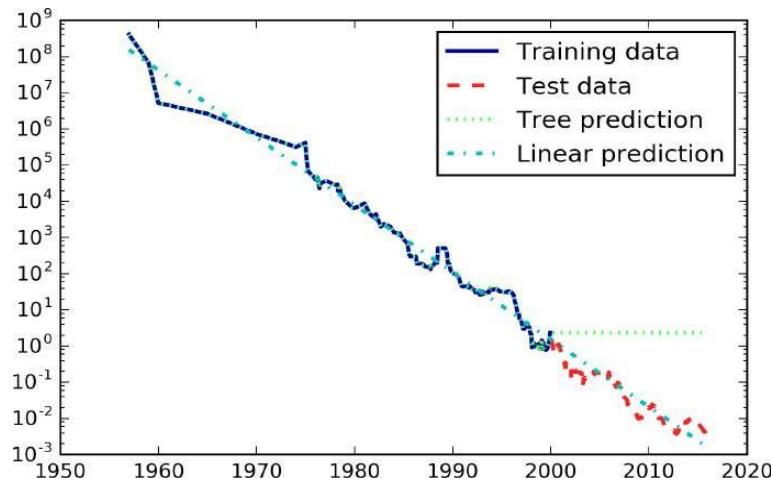
Gambar 2,32, dibuat di sini, membandingkan prediksi pohon keputusan dan model regresi linier dengan kebenaran dasar:

In[67]:

```

plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.legend()

```



**Gambar 2.32** Perbandingan prediksi yang dibuat oleh model linier dan prediksi yang dibuat oleh pohon regresi pada data harga RAM

Perbedaan antara model cukup mencolok. Model linier mendekati data dengan garis, seperti yang kita ketahui. Garis ini memberikan perkiraan yang cukup baik untuk data pengujian (tahun-tahun setelah tahun 2000), sambil menutupi beberapa variasi yang lebih baik dalam data pelatihan maupun pengujian. Model pohon, di sisi lain, membuat prediksi sempurna pada data pelatihan; kami tidak membatasi kerumitan pohon, jadi ia mempelajari seluruh dataset dengan hati. Namun, begitu kita meninggalkan rentang data yang datanya dimiliki model, model terus memprediksi titik terakhir yang diketahui. Pohon tidak memiliki kemampuan untuk menghasilkan respons "baru", di luar apa yang terlihat dalam data pelatihan. Kekurangan ini berlaku untuk semua model berdasarkan pohon.

### Kekuatan, kelemahan, dan parameter

Seperti dibahas sebelumnya, parameter yang mengontrol kompleksitas model dalam pohon keputusan adalah parameter pra-pemangkas yang menghentikan pembangunan

pohon sebelum sepenuhnya dikembangkan. Biasanya, memilih salah satu strategi pra-pemangkasan—mengatur `max_depth`, `max_leaf_nodes`, atau `min_samples_leaf`—cukup untuk mencegah *overfitting*.

Pohon keputusan memiliki dua keunggulan dibandingkan banyak algoritme yang telah kita bahas sejauh ini: model yang dihasilkan dapat dengan mudah divisualisasikan dan dipahami oleh orang yang tidak ahli (setidaknya untuk pohon yang lebih kecil), dan algoritme sama sekali tidak berubah terhadap penskalaan data. Karena setiap fitur diproses secara terpisah, dan kemungkinan pemisahan data tidak bergantung pada penskalaan, tidak ada prapemrosesan seperti normalisasi atau standarisasi fitur yang diperlukan untuk algoritme pohon keputusan. Secara khusus, pohon keputusan bekerja dengan baik ketika Anda memiliki fitur yang berada pada skala yang sama sekali berbeda, atau campuran fitur biner dan kontinu.

Kelemahan utama dari pohon keputusan adalah bahwa bahkan dengan penggunaan pra-pemangkasan, mereka cenderung *overfit* dan memberikan kinerja generalisasi yang buruk. Oleh karena itu, di sebagian besar aplikasi, metode ensemble yang kita bahas selanjutnya biasanya digunakan sebagai pengganti pohon keputusan tunggal.

### **Ensambel Pohon Keputusan**

Ensemble adalah metode yang menggabungkan beberapa model pembelajaran mesin untuk membuat model yang lebih kuat. Ada banyak model dalam literatur pembelajaran mesin yang termasuk dalam kategori ini, tetapi ada dua model ensemble yang telah terbukti efektif pada berbagai kumpulan data untuk klasifikasi dan regresi, yang keduanya menggunakan pohon keputusan sebagai blok bangunannya: acak hutan dan pohon keputusan yang didorong oleh gradien.

### **Hutan acak**

Seperti yang baru saja kita amati, kelemahan utama dari pohon keputusan adalah bahwa mereka cenderung terlalu cocok dengan data pelatihan. Hutan acak adalah salah satu cara untuk mengatasi masalah ini. Hutan acak pada dasarnya adalah kumpulan pohon keputusan, di mana setiap pohon sedikit berbeda dari yang lain. Gagasan di balik hutan acak adalah bahwa setiap pohon mungkin melakukan pekerjaan prediksi yang relatif baik, tetapi kemungkinan akan terlalu banyak memuat sebagian data. Jika kita membangun banyak pohon, yang semuanya bekerja dengan baik dan *overfit* dengan cara yang berbeda, kita dapat mengurangi jumlah *overfitting* dengan merata-ratakan hasilnya. Pengurangan *overfitting* ini, sambil mempertahankan kekuatan prediksi pohon, dapat ditunjukkan dengan menggunakan matematika yang ketat.

Untuk mengimplementasikan strategi ini, kita perlu membangun banyak pohon keputusan. Setiap pohon harus melakukan pekerjaan yang dapat diterima untuk memprediksi target, dan juga harus berbeda dari pohon lainnya. Hutan acak mendapatkan namanya dari menyuntikan keacakan ke dalam bangunan pohon untuk memastikan setiap pohon berbeda. Ada dua cara di mana pohon-pohon di hutan acak diacak: dengan memilih titik data yang digunakan untuk membangun pohon dan dengan memilih fitur di setiap tes split. Mari kita masuk ke proses ini secara lebih rinci.

Membangun hutan acak. Untuk membangun model hutan acak, Anda perlu memutuskan jumlah pohon yang akan dibangun (parameter `n_estimators` dari `RandomForestRegressor` atau `RandomForestClassifier`). Katakanlah kita ingin membangun 10 pohon. Pohon-pohon ini akan dibangun sepenuhnya secara independen satu sama lain, dan algoritme akan membuat pilihan acak yang berbeda untuk setiap pohon untuk memastikan pohon-pohnya berbeda. Untuk membangun pohon, pertama-tama kita mengambil apa yang disebut sampel bootstrap dari data kita. Artinya, dari titik data `n_samples` kami, kami berulang kali menggambar contoh secara acak dengan penggantian (artinya sampel yang sama dapat diambil beberapa kali), `n_samples` kali. Ini akan membuat kumpulan data sebesar kumpulan data asli, tetapi beberapa titik data akan hilang darinya (kira-kira sepertiga), dan beberapa akan berulang.

Sebagai ilustrasi, katakanlah kita ingin membuat contoh bootstrap dari daftar `['a', 'b', 'c', 'd']`. Contoh bootstrap yang mungkin adalah `['b', 'd', 'd', 'c']`. Contoh lain yang mungkin adalah `['d', 'a', 'd', 'a']`.

Selanjutnya, pohon keputusan dibangun berdasarkan dataset yang baru dibuat ini. Namun, algoritma yang kami jelaskan untuk pohon keputusan sedikit dimodifikasi. Alih-alih mencari tes terbaik untuk setiap node, di setiap node, algoritme secara acak memilih subset fitur, dan mencari kemungkinan tes terbaik yang melibatkan salah satu fitur ini. Jumlah fitur yang dipilih dikontrol oleh parameter `max_features`. Pemilihan subset fitur ini diulang secara terpisah di setiap node, sehingga setiap node di pohon dapat membuat keputusan menggunakan subset fitur yang berbeda.

Pengambilan sampel bootstrap mengarah ke setiap pohon keputusan di hutan acak yang dibangun di atas kumpulan data yang sedikit berbeda. Karena pemilihan fitur di setiap node, setiap pemisahan di setiap pohon beroperasi pada subset fitur yang berbeda. Bersama-sama, kedua mekanisme ini memastikan bahwa semua pohon di hutan acak berbeda.

Parameter penting dalam proses ini adalah `max_features`. Jika kita menyetel `max_features` ke `n_features`, itu berarti bahwa setiap pemisahan dapat melihat semua fitur dalam dataset, dan tidak ada keacakan yang akan disuntikkan dalam pemilihan fitur (meskipun keacakan karena bootstrap tetap ada). Jika kami menyetel `max_features` ke 1, itu berarti bahwa pemisahan tidak memiliki pilihan sama sekali pada fitur mana yang akan diuji, dan hanya dapat menelusuri ambang batas yang berbeda untuk fitur yang dipilih secara acak. Oleh karena itu, `max_features` tinggi berarti bahwa pohon-pohon di hutan acak akan sangat mirip, dan mereka akan dapat menyesuaikan data dengan mudah, menggunakan fitur yang paling khas. `Max_features` yang rendah berarti bahwa pohon di hutan acak akan sangat berbeda, dan bahwa setiap pohon mungkin perlu sangat dalam agar sesuai dengan data dengan baik.

Untuk membuat prediksi menggunakan hutan acak, algoritma terlebih dahulu membuat prediksi untuk setiap pohon di hutan. Untuk regresi, kami dapat merata-ratakan hasil ini untuk mendapatkan prediksi akhir kami. Untuk klasifikasi, digunakan strategi "soft voting". Ini berarti setiap algoritma membuat prediksi "lunak", memberikan probabilitas untuk setiap label keluaran yang mungkin. Probabilitas yang diprediksi oleh semua pohon dirata-ratakan, dan kelas dengan probabilitas tertinggi diprediksi.

Menganalisis hutan acak. Mari kita terapkan hutan acak yang terdiri dari lima pohon ke dataset two\_moons yang kita pelajari sebelumnya:

In[68]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=42)

forest = RandomForestClassifier(n_estimators=5, random_state=2)
forest.fit(X_train, y_train)
```

Pohon yang dibangun sebagai bagian dari hutan acak disimpan dalam atribut estimator\_. Mari kita visualisasikan batas keputusan yang dipelajari oleh setiap pohon, bersama dengan prediksi aggregatnya seperti yang dibuat oleh hutan (Gambar 2.33):

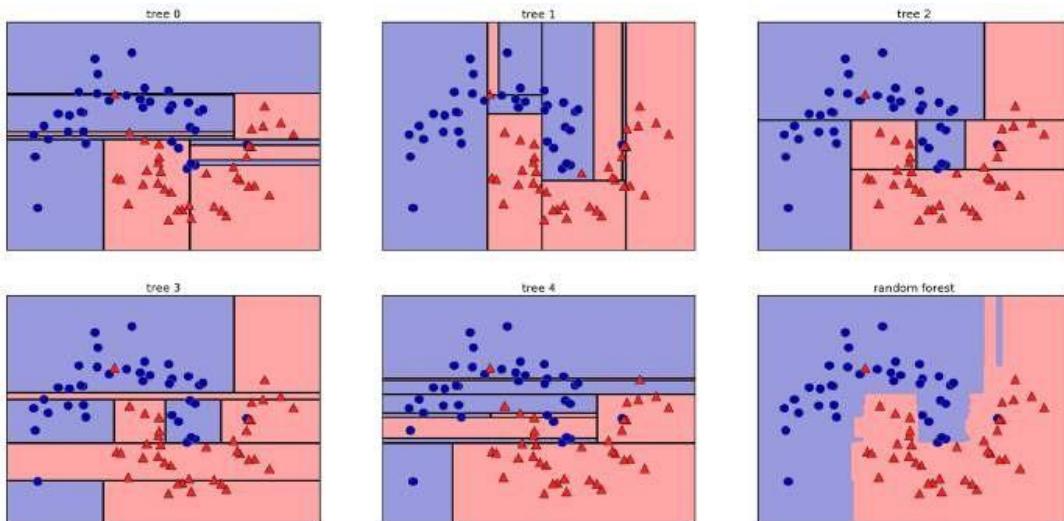
In[69]:

```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("Tree {}".format(i))
    mglearn.plots.plot_tree_partition(X_train, y_train, tree, ax=ax)

    mglearn.plots.plot_2d_separator(forest, X_train, fill=True, ax=axes[-1, -1],
                                    alpha=.4)
    axes[-1, -1].set_title("Random Forest")
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
```

Anda dapat melihat dengan jelas bahwa batasan keputusan yang dipelajari oleh kelima pohon cukup berbeda. Masing-masing dari mereka membuat beberapa kesalahan, karena beberapa titik pelatihan yang diplot di sini sebenarnya tidak termasuk dalam set pelatihan pohon, karena pengambilan sampel bootstrap.

Hutan acak menutupi kurang dari pohon mana pun secara individual, dan memberikan batas keputusan yang jauh lebih intuitif. Dalam aplikasi nyata apa pun, kami akan menggunakan lebih banyak pohon (seringkali ratusan atau ribuan), yang mengarah ke batas yang lebih halus.



**Gambar 2.33** Batas keputusan ditemukan oleh lima pohon keputusan acak dan batas keputusan diperoleh dengan merata-ratakan prediksi probabilitasnya

Sebagai contoh lain, mari kita terapkan hutan acak yang terdiri dari 100 pohon pada dataset Kanker Payudara:

In[70]:

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

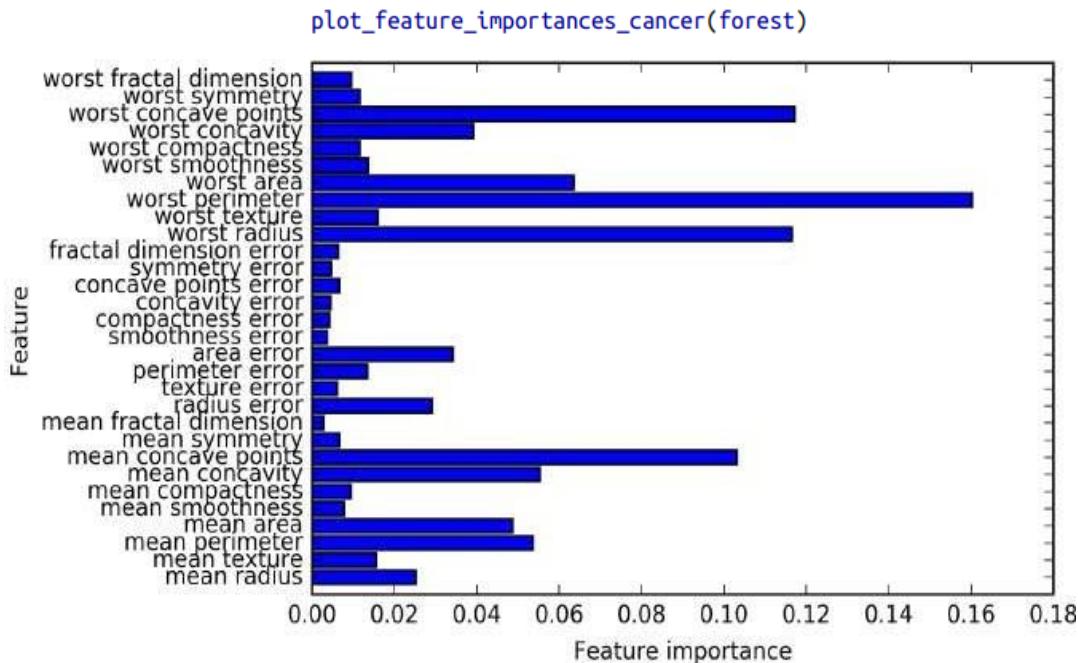
Out[70]:

```
Accuracy on training set: 1.000
Accuracy on test set: 0.972
```

Hutan acak memberi kita akurasi 97%, lebih baik daripada model linier atau pohon keputusan tunggal, tanpa menyetel parameter apa pun. Kita dapat menyesuaikan pengaturan max\_features, atau menerapkan pra-pemangkasan seperti yang kita lakukan untuk pohon keputusan tunggal. Namun, seringkali parameter default dari hutan acak sudah bekerja dengan cukup baik.

Sama halnya dengan pohon keputusan, hutan acak menyediakan fitur penting, yang dihitung dengan menggabungkan fitur penting di atas pohon di hutan. Biasanya, fitur penting yang disediakan oleh hutan acak lebih dapat diandalkan daripada yang disediakan oleh satu pohon. Perhatikan Gambar 2.34.

In[71]:



**Gambar 2.34** Kepentingan fitur dihitung dari hutan acak yang sesuai dengan dataset Kanker Payudara

Seperti yang Anda lihat, hutan acak memberikan kepentingan yang bukan nol pada lebih banyak fitur daripada pohon tunggal. Sama halnya dengan pohon keputusan tunggal, hutan acak juga memberikan banyak kepentingan pada fitur “radius terburuk”, tetapi sebenarnya memilih “perimeter terburuk” untuk menjadi fitur yang paling informatif secara keseluruhan. Keacakan dalam membangun hutan acak memaksa algoritme untuk mempertimbangkan banyak kemungkinan penjelasan, hasilnya adalah bahwa hutan acak menangkap gambaran data yang jauh lebih luas daripada satu pohon.

### **Kekuatan, kelemahan, dan parameter.**

Hutan acak untuk regresi dan klasifikasi saat ini merupakan salah satu metode pembelajaran mesin yang paling banyak digunakan. Mereka sangat kuat, sering bekerja dengan baik tanpa penyetelan parameter yang berat, dan tidak memerlukan penskalaan data.

Pada dasarnya, hutan acak berbagi semua manfaat pohon keputusan, sambil menutupi beberapa kekurangannya. Salah satu alasan untuk tetap menggunakan pohon keputusan adalah jika Anda membutuhkan representasi ringkas dari proses pengambilan keputusan. Pada dasarnya tidak mungkin untuk menafsirkan puluhan atau ratusan pohon secara rinci, dan pohon di hutan acak cenderung lebih dalam daripada pohon keputusan (karena penggunaan subset fitur). Oleh karena itu, jika Anda perlu meringkas pembuatan prediksi secara visual kepada yang bukan ahli, satu pohon keputusan mungkin merupakan pilihan yang lebih baik. Sementara membangun hutan acak pada kumpulan data besar mungkin agak memakan waktu, dapat diparalelkan di beberapa inti CPU dalam komputer dengan mudah. Jika Anda menggunakan prosesor multi-core (seperti yang dilakukan hampir semua komputer modern), Anda dapat menggunakan parameter `n_jobs` untuk menyesuaikan jumlah core yang akan digunakan. Menggunakan lebih banyak inti CPU akan menghasilkan percepatan linier (menggunakan dua inti, pelatihan hutan acak akan dua kali lebih cepat), tetapi menentukan `n_jobs` lebih besar dari jumlah inti tidak akan membantu. Anda dapat mengatur `n_jobs=-1` untuk menggunakan semua inti di komputer Anda.

Anda harus ingat bahwa hutan acak, berdasarkan sifatnya, adalah acak, dan menyetel status acak yang berbeda (atau tidak menyetel status acak sama sekali) dapat mengubah model yang dibangun secara drastis. Semakin banyak pohon yang ada di hutan, semakin kuat ia melawan pilihan keadaan acak. Jika Anda ingin mendapatkan hasil yang dapat direproduksi, penting untuk memperbaiki `random_state`.

Hutan acak cenderung tidak berkinerja baik pada data yang jarang dan berdimensi sangat tinggi, seperti data teks. Untuk jenis data ini, model linier mungkin lebih tepat. Hutan acak biasanya bekerja dengan baik bahkan pada kumpulan data yang sangat besar, dan pelatihan dapat dengan mudah diparalelkan pada banyak inti CPU dalam komputer yang kuat. Namun, hutan acak membutuhkan lebih banyak memori dan lebih lambat untuk dilatih dan diprediksi daripada model linier. Jika waktu dan memori penting dalam suatu aplikasi, mungkin masuk akal untuk menggunakan model linier sebagai gantinya.

Parameter penting yang harus disesuaikan adalah `n_estimators`, `max_features`, dan kemungkinan opsi pra-pemangkasan seperti `max_depth`. Untuk `n_estimator`, lebih besar selalu lebih baik. Rata-rata lebih banyak pohon akan menghasilkan ansambel yang lebih kuat dengan mengurangi overfitting. Namun, ada hasil yang semakin berkurang, dan lebih banyak

pohon membutuhkan lebih banyak memori dan lebih banyak waktu untuk berlatih. Aturan praktis yang umum adalah membangun “sebanyak yang Anda punya waktu/ingatan.”

Seperti dijelaskan sebelumnya, `max_features` menentukan seberapa acak setiap pohon, dan `max_features` yang lebih kecil mengurangi overfitting. Secara umum, ini adalah aturan praktis yang baik untuk menggunakan nilai default: `max_features=sqrt(n_features)` untuk klasifikasi dan `max_features=log2(n_features)` untuk regresi. Menambahkan `max_features` atau `max_leaf_nodes` terkadang dapat meningkatkan kinerja. Ini juga dapat secara drastis mengurangi persyaratan ruang dan waktu untuk pelatihan dan prediksi.

### **Pohon regresi yang didorong gradien (mesin penambah gradien)**

Pohon regresi yang didorong gradien adalah metode ensemble lain yang menggabungkan beberapa pohon keputusan untuk membuat model yang lebih kuat. Terlepas dari "regresi" dalam namanya, model ini dapat digunakan untuk regresi dan klasifikasi. Berbeda dengan pendekatan hutan acak, peningkatan gradien bekerja dengan membangun pohon secara serial, di mana setiap pohon mencoba memperbaiki kesalahan yang sebelumnya. Secara default, tidak ada pengacakan dalam pohon regresi yang didorong oleh gradien; sebagai gantinya, pra-pemangkasan yang kuat digunakan. Pohon yang didorong gradien sering kali menggunakan pohon yang sangat dangkal, dengan kedalaman satu hingga lima, yang membuat model lebih kecil dalam hal memori dan membuat prediksi lebih cepat.

Gagasan utama di balik peningkatan gradien adalah menggabungkan banyak model sederhana (dalam konteks ini dikenal sebagai pembelajar yang lemah), seperti pohon yang dangkal. Setiap pohon hanya dapat memberikan prediksi yang baik pada sebagian data, dan semakin banyak pohon ditambahkan untuk meningkatkan kinerja secara iteratif.

Pohon yang didorong gradien sering kali merupakan entri pemenang dalam kompetisi pembelajaran mesin, dan banyak digunakan di industri. Mereka umumnya sedikit lebih sensitif terhadap pengaturan parameter daripada hutan acak, tetapi dapat memberikan akurasi yang lebih baik jika parameter diatur dengan benar.

Terlepas dari pra-pemangkasan dan jumlah pohon dalam ensemble, parameter penting lain dari peningkatan gradien adalah `learning_rate`, yang mengontrol seberapa kuat setiap pohon mencoba memperbaiki kesalahan pohon sebelumnya. Tingkat pembelajaran yang lebih tinggi berarti setiap pohon dapat membuat koreksi yang lebih kuat, memungkinkan model yang lebih kompleks. Menambahkan lebih banyak pohon ke ensemble, yang dapat dicapai dengan meningkatkan `n_estimators`, juga meningkatkan kompleksitas model, karena model memiliki lebih banyak peluang untuk memperbaiki kesalahan pada set pelatihan.

Berikut adalah contoh penggunaan `GradientBoostingClassifier` pada dataset Kanker Payudara. Secara default, 100 pohon dengan kedalaman maksimum 3 dan kecepatan pembelajaran 0,1 digunakan:

In[72]:

```
from sklearn.ensemble import GradientBoostingClassifier

X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbdt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbdt.score(X_test, y_test)))
```

Out[72]:

```
Accuracy on training set: 1.000
Accuracy on test set: 0.958
```

Karena akurasi set pelatihan adalah 100%, kami cenderung melakukan overfitting. Untuk mengurangi overfitting, kita dapat menerapkan pra-pangkasan yang lebih kuat dengan membatasi kedalaman maksimum atau menurunkan kecepatan belajar:

In[73]:

```
gbdt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbdt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbdt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbdt.score(X_test, y_test)))
```

Out[73]:

```
Accuracy on training set: 0.991
Accuracy on test set: 0.972
```

In[74]:

```
gbdt = GradientBoostingClassifier(random_state=0, learning_rate=0.01)
gbdt.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(gbdt.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gbdt.score(X_test, y_test)))
```

Out[74]:

```
Accuracy on training set: 0.988
Accuracy on test set: 0.965
```

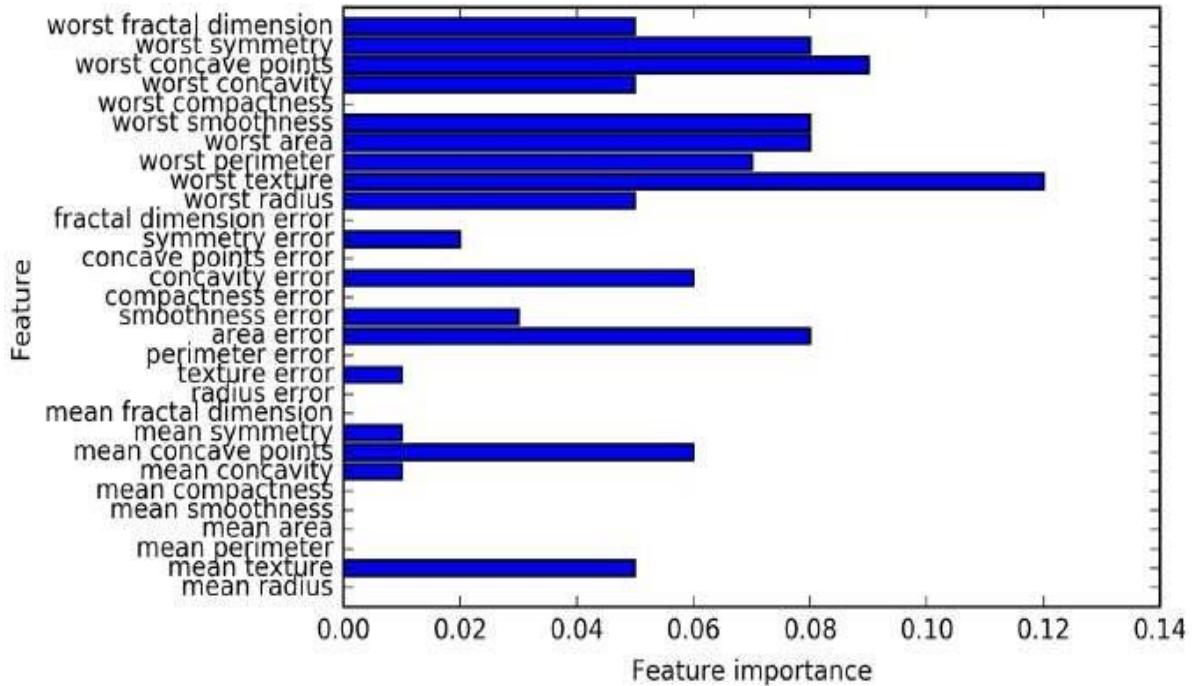
Kedua metode penurunan kompleksitas model mengurangi akurasi set pelatihan, seperti yang diharapkan. Dalam hal ini, menurunkan kedalaman maksimum pohon memberikan peningkatan model yang signifikan, sementara menurunkan tingkat pembelajaran hanya sedikit meningkatkan kinerja generalisasi.

Untuk model berbasis pohon keputusan lainnya, kami dapat kembali memvisualisasikan fitur penting untuk mendapatkan lebih banyak wawasan tentang model kami (Gambar 2.35). Karena kami menggunakan 100 pohon, tidak praktis untuk memeriksa semuanya, meskipun semuanya memiliki kedalaman 1:

In[75]:

```
gbrt = GradientBoostingClassifier(random_state=0, max_depth=1)
gbrt.fit(X_train, y_train)

plot_feature_importances_cancer(gbdt)
```



**Gambar 2.35** Kepentingan fitur dihitung dari pengklasifikasi penambah gradien yang sesuai dengan dataset Kanker Payudara

Kita dapat melihat bahwa pentingnya fitur dari pohon yang didorong gradien agak mirip dengan pentingnya fitur dari hutan acak, meskipun peningkatan gradien sepenuhnya mengabaikan beberapa fitur.

Karena peningkatan gradien dan hutan acak berkinerja baik pada jenis data yang serupa, pendekatan umum adalah pertama-tama mencoba hutan acak, yang bekerja dengan cukup kuat. Jika hutan acak bekerja dengan baik tetapi waktu prediksi sangat mahal, atau penting untuk memeras persentase akurasi terakhir dari model pembelajaran mesin, beralih ke peningkatan gradien sering kali membantu.

Jika Anda ingin menerapkan peningkatan gradien ke masalah skala besar, mungkin ada baiknya melihat ke paket xgboost dan antarmuka Python-nya, yang pada saat penulisan lebih cepat (dan terkadang lebih mudah untuk disetel) daripada scikit-learn implementasi peningkatan gradien pada banyak kumpulan data.

Kekuatan, kelemahan, dan parameter. Pohon keputusan yang didorong gradien adalah salah satu model yang paling kuat dan banyak digunakan untuk pembelajaran yang diawasi. Kelemahan utama mereka adalah bahwa mereka memerlukan penyetelan parameter yang cermat dan mungkin membutuhkan waktu lama untuk dilatih. Mirip dengan model berbasis pohon lainnya, algoritma bekerja dengan baik tanpa penskalaan dan pada campuran fitur

biner dan kontinu. Seperti model berbasis pohon lainnya, model ini juga sering tidak berfungsi dengan baik pada data sparse berdimensi tinggi.

Parameter utama model pohon yang didorong gradien adalah jumlah pohon, `n_estimator`, dan `learning_rate`, yang mengontrol sejauh mana setiap pohon diperbolehkan untuk memperbaiki kesalahan pohon sebelumnya. Kedua parameter ini sangat saling berhubungan, karena `learning_rate` yang lebih rendah berarti bahwa lebih banyak pohon diperlukan untuk membangun model dengan kompleksitas yang serupa. Berbeda dengan hutan acak, di mana nilai `n_estimator` yang lebih tinggi selalu lebih baik, peningkatan `n_estimator` dalam peningkatan gradien mengarah ke model yang lebih kompleks, yang dapat menyebabkan overfitting. Praktik yang umum adalah menyesuaikan `n_estimator` tergantung pada waktu dan anggaran memori, dan kemudian mencari `tingkat_pembelajaran` yang berbeda.

Parameter penting lainnya adalah `max_depth` (atau alternatifnya `max_leaf_nodes`), untuk mengurangi kompleksitas setiap pohon. Biasanya `max_depth` diatur sangat rendah untuk model yang didorong gradien, seringkali tidak lebih dari lima split.

## 2.10 MESIN VEKTOR DUKUNGAN KERNELIZED

Jenis model terawasi berikutnya yang akan kita bahas adalah mesin vektor pendukung kernel. Kami mengeksplorasi penggunaan mesin vektor pendukung linier untuk klasifikasi dalam “Model linier untuk klasifikasi” di halaman 56. Mesin vektor pendukung berkernel (sering disebut sebagai SVM) adalah ekstensi yang memungkinkan model yang lebih kompleks yang tidak didefinisikan secara sederhana oleh hyperplanes di ruang input. Meskipun ada mesin vektor pendukung untuk klasifikasi dan regresi, kami akan membatasi diri pada kasus klasifikasi, seperti yang diterapkan di SVC. Konsep serupa berlaku untuk mendukung regresi vektor, seperti yang diterapkan di SVR.

Matematika di balik mesin vektor dukungan kernel sedikit terlibat, dan berada di luar cakupan buku ini. Anda dapat menemukan detailnya di Bab 1 dari Hastie, Tibshirani, dan The Elements of Statistical Learning karya Friedman. Namun, kami akan mencoba memberi Anda gambaran tentang ide di balik metode ini.

### Model linier dan fitur nonlinier

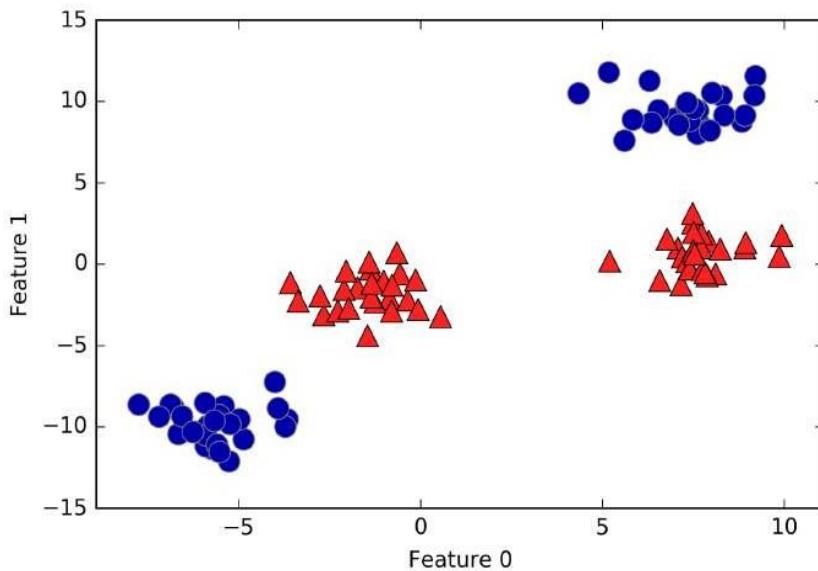
Seperti yang Anda lihat pada Gambar 2-15, model linier bisa sangat membatasi ruang dimensi rendah, karena garis dan hyperplane memiliki fleksibilitas terbatas. Salah satu cara untuk membuat model linier lebih fleksibel adalah dengan menambahkan lebih banyak fitur—misalnya, dengan menambahkan interaksi atau polinomial dari fitur input.

Mari kita lihat kumpulan data sintetik yang kita gunakan di “Kepentingan fitur dalam pohon” di halaman 77 (lihat Gambar 2-39):

In[76]:

```
X, y = make_blobs(centers=4, random_state=8)
y = y % 2

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 2.36** Dataset klasifikasi dua kelas di mana kelas tidak dapat dipisahkan secara linier

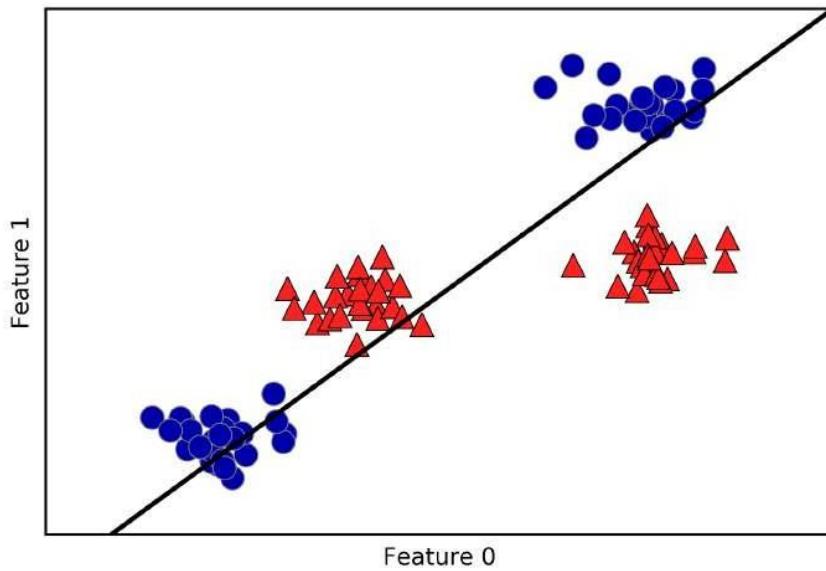
Model linier untuk klasifikasi hanya dapat memisahkan titik menggunakan garis, dan tidak akan dapat melakukan pekerjaan yang sangat baik pada kumpulan data ini (lihat Gambar 2.37):

In[77]:

```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Sekarang mari kita perluas kumpulan fitur input, misalnya dengan menambahkan  $\text{fitur1}^{** 2}$ , kuadrat dari fitur kedua, sebagai fitur baru. Alih-alih mewakili setiap titik data sebagai titik dua dimensi, ( $\text{fitur0}, \text{fitur1}$ ), sekarang kita merepresentasikannya sebagai titik tiga dimensi, ( $\text{fitur0}, \text{fitur1}, \text{fitur1}^{** 2}$ ). Representasi baru ini diilustrasikan dalam Gambar 2.38 dalam plot sebar tiga dimensi:

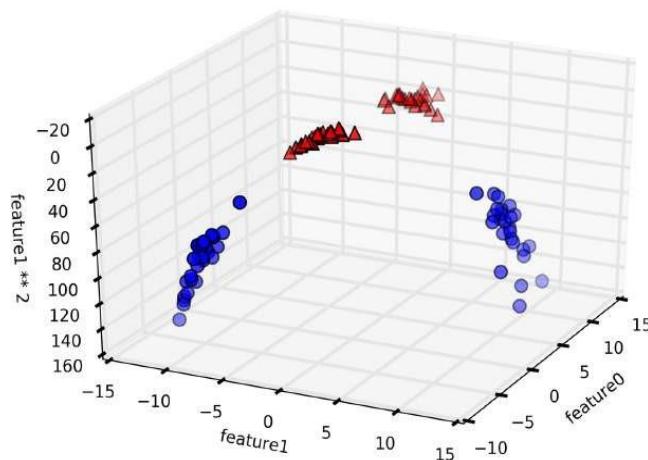


**Gambar 2.37** Batas keputusan ditemukan oleh SVM linier

In[78]:

```
# add the squared first feature
X_new = np.hstack([X, X[:, 1:] ** 2])

from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure()
# visualize in 3D
ax = Axes3D(figure, elev=-152, azim=-26)
# plot first all the points with y == 0, then all with y == 1
mask = y == 0
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b',
           cmap=mlearn.cm2, s=60)
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
           cmap=mlearn.cm2, s=60)
ax.set_xlabel("feature0")
ax.set_ylabel("feature1")
ax.set_zlabel("feature1 ** 2")
```



**Gambar 2.38** Perluasan dataset yang ditunjukkan pada *Gambar 2.37*, dibuat dengan menambahkan fitur ketiga yang berasal dari fitur1

Dalam representasi data yang baru, sekarang memang dimungkinkan untuk memisahkan dua kelas menggunakan model linier, bidang tiga dimensi. Kami dapat mengkonfirmasi ini dengan memasang model linier ke data yang ditambah (lihat Gambar 2.39):

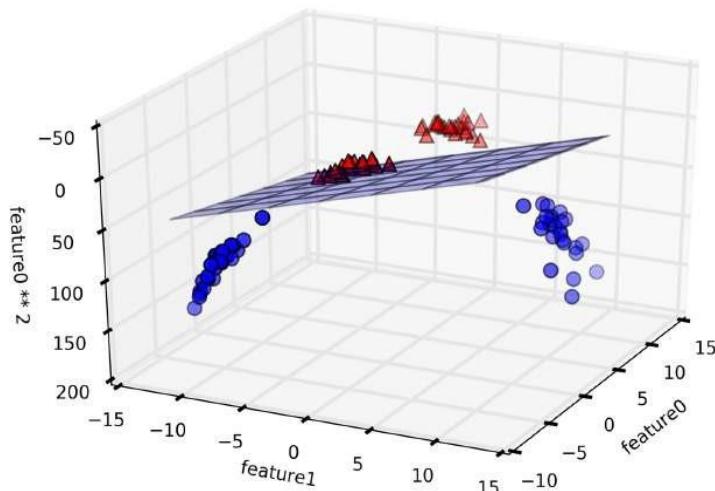
In[79]:

```
linear_svm_3d = LinearSVC().fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_

# show linear decision boundary
figure = plt.figure()
ax = Axes3D(figure, elev=-152, azim=-26)
xx = np.linspace(X_new[:, 0].min() - 2, X_new[:, 0].max() + 2, 50)
yy = np.linspace(X_new[:, 1].min() - 2, X_new[:, 1].max() + 2, 50)

XX, YY = np.meshgrid(xx, yy)
ZZ = (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b',
           cmap=mlearn.cm2, s=60)
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
           cmap=mlearn.cm2, s=60)

ax.set_xlabel("feature0")
ax.set_ylabel("feature1")
ax.set_zlabel("feature0 ** 2")
```



**Gambar 2.39** Batas keputusan yang ditemukan oleh SVM linier pada kumpulan data tiga dimensi yang diperluas

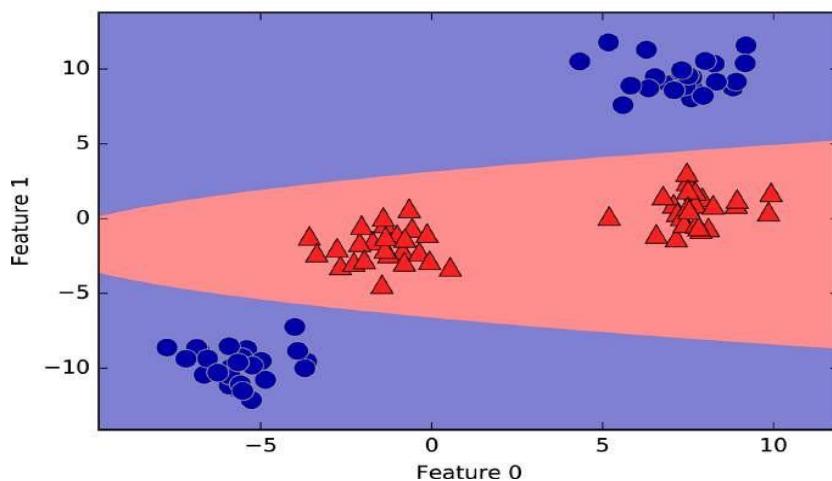
Sebagai fungsi dari fitur asli, model SVM linier sebenarnya tidak linier lagi. Ini bukan garis, tetapi lebih seperti elips, seperti yang Anda lihat dari plot yang dibuat di sini (Gambar 2.40):

In[80]:

```
ZZ = YY ** 2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel(), YY.ravel(), ZZ.ravel()])
plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(), 0, dec.max()],
             cmap=mlearn.cm2, alpha=0.5)
mlearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

### Trik kernel

Pelajaran di sini adalah bahwa menambahkan fitur nonlinier ke representasi data kami dapat membuat model linier jauh lebih kuat. Namun, seringkali kita tidak tahu fitur mana yang harus ditambahkan, dan menambahkan banyak fitur (seperti semua kemungkinan interaksi dalam ruang fitur 100 dimensi) mungkin membuat komputasi menjadi sangat mahal. Untungnya, ada trik matematika cerdas yang memungkinkan kita mempelajari pengklasifikasi dalam ruang dimensi yang lebih tinggi tanpa benar-benar menghitung representasi baru yang mungkin sangat besar. Ini dikenal sebagai trik kernel, dan bekerja dengan menghitung secara langsung jarak (lebih tepatnya, produk skalar) dari titik data untuk representasi fitur yang diperluas, tanpa pernah benar-benar menghitung ekspansi.



**Gambar 2.40** Batas keputusan dari Gambar 2.39 sebagai fungsi dari dua fitur asli

Ada dua cara untuk memetakan data Anda ke dalam ruang berdimensi lebih tinggi yang biasanya digunakan dengan mesin vektor pendukung: kernel polinomial, yang menghitung semua polinomial yang mungkin hingga derajat tertentu dari fitur asli (seperti fitur1 \*\*2 \* fitur2 \*\*5); dan kernel fungsi basis radial (RBF), juga dikenal sebagai kernel Gaussian. Kernel Gaussian sedikit lebih sulit untuk dijelaskan, karena sesuai dengan ruang fitur dimensi tak terbatas. Salah satu cara untuk menjelaskan kernel Gaussian adalah dengan mempertimbangkan semua polinomial yang mungkin dari semua derajat, tetapi pentingnya fitur berkurang untuk derajat yang lebih tinggi.

Namun, dalam praktiknya, detail matematis di balik kernel SVM tidak begitu penting, dan bagaimana SVM dengan kernel RBF membuat keputusan dapat diringkas dengan cukup mudah—kita akan melakukannya di bagian berikutnya.

### Memahami SVM

Selama pelatihan, SVM mempelajari betapa pentingnya setiap titik data pelatihan untuk mewakili batas keputusan antara dua kelas. Biasanya hanya sebagian dari poin pelatihan yang penting untuk mendefinisikan batas keputusan: yang terletak di perbatasan antara kelas. Ini disebut support vector dan beri nama mesin support vector tersebut.

Untuk membuat prediksi untuk titik baru, jarak ke masing-masing vektor pendukung diukur. Keputusan klasifikasi dibuat berdasarkan jarak ke vektor pendukung, dan pentingnya vektor pendukung yang dipelajari selama pelatihan (disimpan dalam atribut `dual_coef_` dari SVC). Jarak antara titik data diukur dengan kernel Gaussian:

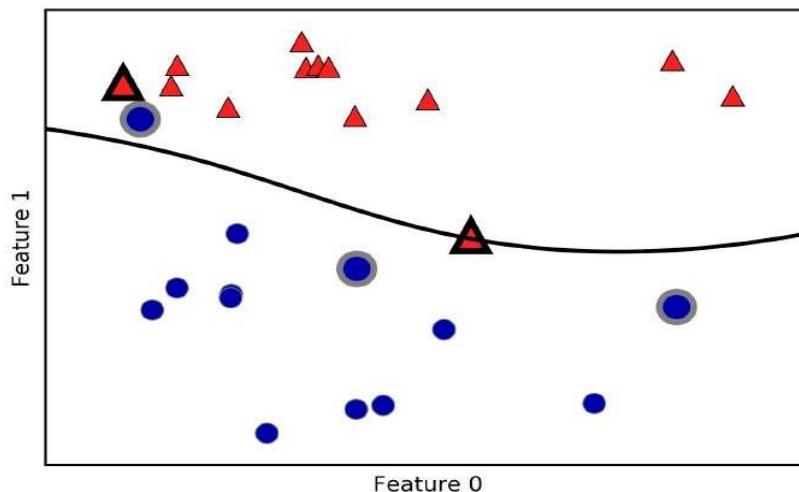
$$k_{\text{rbf}}(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$$

Di sini,  $x_1$  dan  $x_2$  adalah titik data,  $\|x_1 - x_2\|$  menunjukkan jarak Euclidean, dan  $\gamma$  (gamma) adalah parameter yang mengontrol lebar kernel Gaussian.

Gambar 2.41 menunjukkan hasil pelatihan mesin vektor pendukung pada dataset kelas dua dimensi. Batas keputusan ditampilkan dalam warna hitam, dan vektor pendukung adalah titik yang lebih besar dengan garis besar yang lebar. Kode berikut membuat plot ini dengan melatih SVM pada dataset forge:

In[81]:

```
from sklearn.svm import SVC
X, y = mglearn.tools.make_handcrafted_dataset()
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
mglearn.plots.plot_2d_separator(svm, X, eps=.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
# plot support vectors
sv = svm.support_vectors_
# class labels of support vectors are given by the sign of the dual coefficients
sv_labels = svm.dual_coef_.ravel() > 0
mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15, markeredgewidth=3)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 2.41** Batas keputusan dan vektor dukungan yang ditemukan oleh SVM dengan kernel RBF

Dalam hal ini, SVM menghasilkan batas yang sangat halus dan nonlinier (bukan garis lurus). Kami menyesuaikan dua parameter di sini: parameter  $C$  dan parameter  $\gamma$ , yang sekarang akan kita bahas secara rinci.

### Menyetel parameter SVM

Parameter  $\gamma$  adalah yang ditunjukkan dalam rumus yang diberikan di bagian sebelumnya, yang mengontrol lebar kernel Gaussian. Ini menentukan skala dari apa artinya

poin menjadi berdekatan. Parameter C adalah parameter regularisasi, mirip dengan yang digunakan dalam model linier. Ini membatasi pentingnya setiap poin (atau lebih tepatnya, dual\_coef\_ mereka).

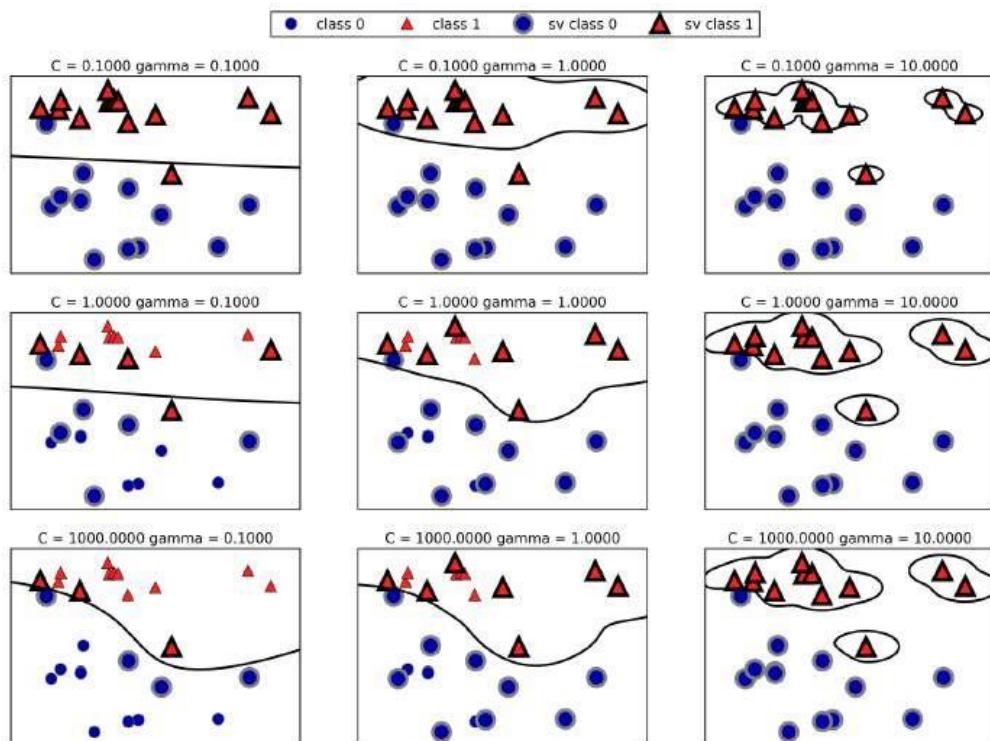
Mari kita lihat apa yang terjadi ketika kita memvariasikan parameter ini (Gambar 2.42):

In[82]:

```
fig, axes = plt.subplots(3, 3, figsize=(15, 10))

for ax, C in zip(axes, [-1, 0, 3]):
    for a, gamma in zip(ax, range(-1, 2)):
        mlearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=a)

axes[0, 0].legend(["class 0", "class 1", "sv class 0", "sv class 1"],
                  ncol=4, loc=(.9, 1.2))
```



**Gambar 2.42** Batas keputusan dan vektor pendukung untuk pengaturan yang berbeda dari parameter C dan gamma

Dari kiri ke kanan, kami meningkatkan nilai parameter gamma dari 0,1 menjadi 10. Gamma kecil berarti radius besar untuk kernel Gaussian, yang berarti banyak titik dianggap berdekatan. Hal ini tercermin dalam batas-batas keputusan yang sangat halus di sebelah kiri, dan batas-batas yang lebih fokus pada satu titik lebih jauh ke kanan. Nilai gamma yang rendah berarti batas keputusan akan bervariasi secara perlahan, yang menghasilkan model dengan kompleksitas rendah, sedangkan nilai gamma yang tinggi menghasilkan model yang lebih kompleks.

Dari atas ke bawah, kami meningkatkan parameter C dari 0,1 menjadi 1000. Seperti pada model linier, C kecil berarti model yang sangat terbatas, di mana setiap titik data hanya dapat memiliki pengaruh yang sangat terbatas. Anda dapat melihat bahwa di kiri atas batas

keputusan terlihat hampir linier, dengan titik-titik yang salah diklasifikasikan hampir tidak memiliki pengaruh pada garis. Meningkatkan C, seperti yang ditunjukkan di kanan bawah, memungkinkan titik-titik ini memiliki pengaruh yang lebih kuat pada model dan membuat batas keputusan ditekuk untuk mengklasifikasikannya dengan benar.

Mari kita terapkan kernel RBF SVM ke dataset Kanker Payudara. Secara default, C=1 dan gamma=1/n\_features:

**In[83]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

svc = SVC()
svc.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

**Out[83]:**

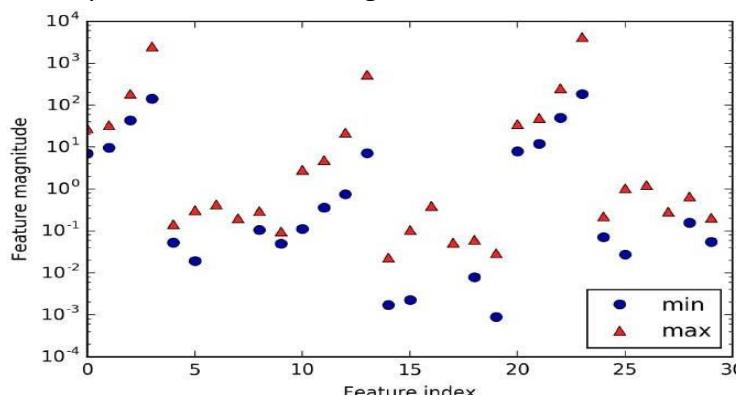
```
Accuracy on training set: 1.00
Accuracy on test set: 0.63
```

Model overfits cukup substansial, dengan skor sempurna pada set pelatihan dan akurasi hanya 63% pada set tes. Meskipun SVM sering kali berkinerja cukup baik, mereka sangat sensitif terhadap pengaturan parameter dan penskalaan data. Secara khusus, mereka membutuhkan semua fitur untuk bervariasi pada skala yang sama. Mari kita lihat nilai minimum dan maksimum untuk setiap fitur, diplot dalam log-space (Gambar 2.43):

**In[84]:**

```
plt.plot(X_train.min(axis=0), 'o', label="min")
plt.plot(X_train.max(axis=0), '^', label="max")
plt.legend(loc=4)
plt.xlabel("Feature index")
plt.ylabel("Feature magnitude")
plt.yscale("log")
```

Dari plot ini kita dapat menentukan bahwa fitur dalam kumpulan data Kanker Payudara memiliki urutan besarnya yang sangat berbeda. Ini bisa menjadi masalah untuk model lain (seperti model linier), tetapi memiliki efek yang menghancurkan untuk kernel SVM. Mari kita periksa beberapa cara untuk menangani masalah ini.



**Gambar 2.43** Rentang fitur untuk dataset Kanker Payudara (perhatikan bahwa sumbu y memiliki skala logaritmik)

## Memproses data untuk SVM

Salah satu cara untuk mengatasi masalah ini adalah dengan menskalakan ulang setiap fitur sehingga semuanya kira-kira pada skala yang sama. Metode penskalaan ulang yang umum untuk SVM kernel adalah menskalakan data sedemikian rupa sehingga semua fitur berada di antara 0 dan 1. Kita akan melihat bagaimana melakukannya menggunakan metode prapemrosesan MinMaxScaler di Bab 3, di mana kita akan memberikan rincian lebih lanjut. Untuk saat ini, mari lakukan ini "dengan tangan":

**In[85]:**

```
# compute the minimum value per feature on the training set
min_on_training = X_train.min(axis=0)
# compute the range of each feature (max - min) on the training set
range_on_training = (X_train - min_on_training).max(axis=0)

# subtract the min, and divide by range
# afterward, min=0 and max=1 for each feature
X_train_scaled = (X_train - min_on_training) / range_on_training
print("Minimum for each feature\n{}".format(X_train_scaled.min(axis=0)))
print("Maximum for each feature\n {}".format(X_train_scaled.max(axis=0)))
```

**Out[85]:**

```
Minimum for each feature
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
Maximum for each feature
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

**In[86]:**

```
# use THE SAME transformation on the test set,
# using min and range of the training set (see Chapter 3 for details)
X_test_scaled = (X_test - min_on_training) / range_on_training
```

**In[87]:**

```
svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

**Out[87]:**

```
Accuracy on training set: 0.948
Accuracy on test set: 0.951
```

Menskalakan data membuat perbedaan besar! Sekarang kita sebenarnya berada dalam rezim yang kurang pas, di mana kinerja set pelatihan dan pengujian sangat mirip tetapi kurang mendekati akurasi 100%. Dari sini, kita dapat mencoba meningkatkan C atau gamma agar sesuai dengan model yang lebih kompleks. Sebagai contoh:

**In[88]:**

```
svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

**Out[88]:**

```
Accuracy on training set: 0.988
Accuracy on test set: 0.972
```

Di sini, meningkatkan C memungkinkan kami meningkatkan model secara signifikan, menghasilkan akurasi 97,2%.

### Kekuatan, kelemahan, dan parameter

Mesin vektor dukungan kernel adalah model yang kuat dan berkinerja baik pada berbagai set data. SVM memungkinkan batasan keputusan yang kompleks, bahkan jika data hanya memiliki beberapa fitur. Mereka bekerja dengan baik pada data berdimensi rendah dan tinggi (yaitu, sedikit dan banyak fitur), tetapi tidak berskala sangat baik dengan jumlah sampel. Menjalankan SVM pada data hingga 10.000 sampel mungkin bekerja dengan baik, tetapi bekerja dengan kumpulan data berukuran 100.000 atau lebih dapat menjadi tantangan dalam hal waktu proses dan penggunaan memori.

Kelemahan lain dari SVM adalah bahwa mereka memerlukan pra-pemrosesan data yang cermat dan penyetelan parameter. Inilah sebabnya mengapa, akhir-akhir ini, kebanyakan orang menggunakan model berbasis pohon seperti hutan acak atau peningkatan gradien (yang memerlukan sedikit atau tanpa pra-pemrosesan) di banyak aplikasi. Selain itu, model SVM sulit untuk diperiksa; mungkin sulit untuk memahami mengapa prediksi tertentu dibuat, dan mungkin sulit untuk menjelaskan model kepada orang yang tidak ahli.

Namun, mungkin ada baiknya mencoba SVM, terutama jika semua fitur Anda mewakili pengukuran dalam unit yang serupa (misalnya, semua adalah intensitas piksel) dan mereka berada pada skala yang sama.

Parameter penting dalam SVM kernel adalah parameter regularisasi C, pilihan kernel, dan parameter spesifik kernel. Meskipun kami terutama berfokus pada kernel RBF, pilihan lain tersedia di scikit-learn. Kernel RBF hanya memiliki satu parameter, gamma, yang merupakan kebalikan dari lebar kernel Gaussian. gamma dan C sama-sama mengontrol kompleksitas model, dengan nilai yang besar dalam keduanya menghasilkan model yang lebih kompleks. Oleh karena itu, pengaturan yang baik untuk kedua parameter biasanya berkorelasi kuat, dan C dan gamma harus disesuaikan bersama.

## 2.11 JARINGAN NEURAL (PEMBELAJARAN MENDALAM)

Sebuah keluarga algoritma yang dikenal sebagai jaringan saraf baru-baru ini melihat kebangkitan dengan nama "pembelajaran mendalam." Sementara pembelajaran mendalam menunjukkan harapan besar di banyak aplikasi pembelajaran mesin, algoritme pembelajaran mendalam sering kali disesuaikan dengan sangat hati-hati untuk kasus penggunaan tertentu. Di sini, kita hanya akan membahas beberapa metode yang relatif sederhana, yaitu multilayer

perceptrons untuk klasifikasi dan regresi, yang dapat berfungsi sebagai titik awal untuk metode pembelajaran yang lebih mendalam. Multilayer perceptrons (MLPs) juga dikenal sebagai (vanilla) *feed-forward neural networks*, atau terkadang hanya neural network.

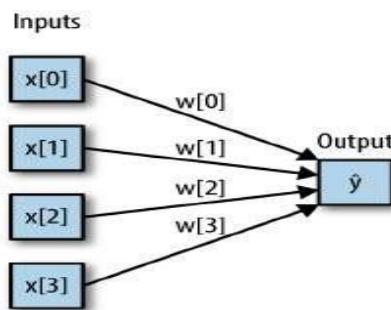
### Model jaringan saraf

MLP dapat dilihat sebagai generalisasi dari model linier yang melakukan beberapa tahap pemrosesan untuk mengambil keputusan. Ingat bahwa prediksi oleh regresi linier diberikan sebagai:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b$$

**In[89]:**

```
display(mglearn.plots.plot_logistic_regression_graph())
```



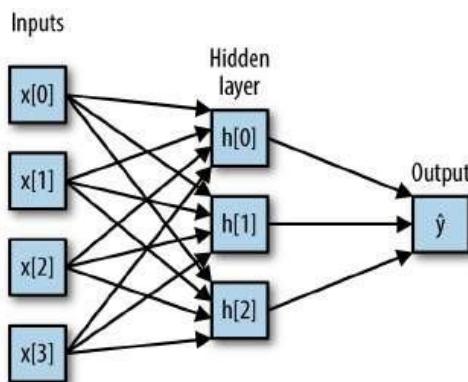
**Gambar 2.44** Visualisasi regresi logistik, di mana fitur input dan prediksi ditampilkan sebagai node, dan koefisiennya adalah koneksi antar node

Di sini, setiap node di sebelah kiri mewakili fitur input, garis penghubung mewakili koefisien yang dipelajari, dan node di sebelah kanan mewakili output, yang merupakan jumlah bobot dari input.

Dalam MLP, proses menghitung jumlah tertimbang ini diulang beberapa kali, pertama-tama menghitung unit tersembunyi yang mewakili langkah pemrosesan menengah, yang lagi-lagi digabungkan menggunakan jumlah berbobot untuk menghasilkan hasil akhir (Gambar 2.45):

**In[90]:**

```
display(mglearn.plots.plot_single_hidden_layer_graph())
```



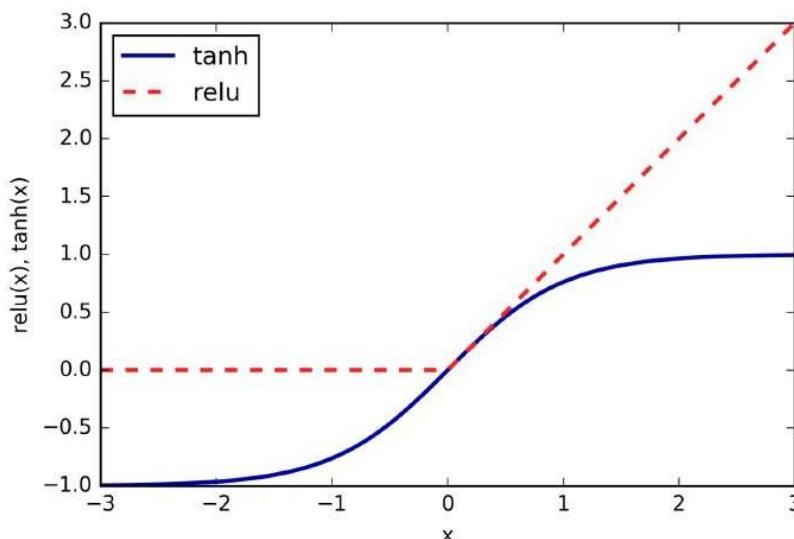
**Gambar 2.45** Ilustrasi perceptron multilayer dengan satu lapisan tersembunyi

Model ini memiliki lebih banyak koefisien (juga disebut bobot) untuk dipelajari: ada satu di antara setiap input dan setiap unit tersembunyi (yang membentuk lapisan tersembunyi), dan satu di antara setiap unit di lapisan tersembunyi dan output.

Menghitung serangkaian jumlah berbobot secara matematis sama dengan menghitung hanya satu jumlah berbobot, jadi untuk membuat model ini benar-benar lebih kuat daripada model linier, kita memerlukan satu trik tambahan. Setelah menghitung jumlah bobot untuk setiap unit tersembunyi, fungsi nonlinier diterapkan pada hasil—biasanya penyebaran nonlinier (juga dikenal sebagai unit linier terkoreksi atau relu) atau tangens hiperbolik (tanh). Hasil dari fungsi ini kemudian digunakan dalam jumlah tertimbang yang menghitung keluaran. Kedua fungsi tersebut divisualisasikan pada Gambar 2-36. Relu memotong nilai di bawah nol, sementara tanh jenuh ke -1 untuk nilai input rendah dan +1 untuk nilai input tinggi. Salah satu fungsi nonlinier memungkinkan jaringan saraf untuk mempelajari fungsi yang jauh lebih rumit daripada yang dapat dilakukan oleh model linier:

In[91]:

```
line = np.linspace(-3, 3, 100)
plt.plot(line, np.tanh(line), label="tanh")
plt.plot(line, np.maximum(line, 0), label="relu")
plt.legend(loc="best")
plt.xlabel("x")
plt.ylabel("relu(x), tanh(x)")
```



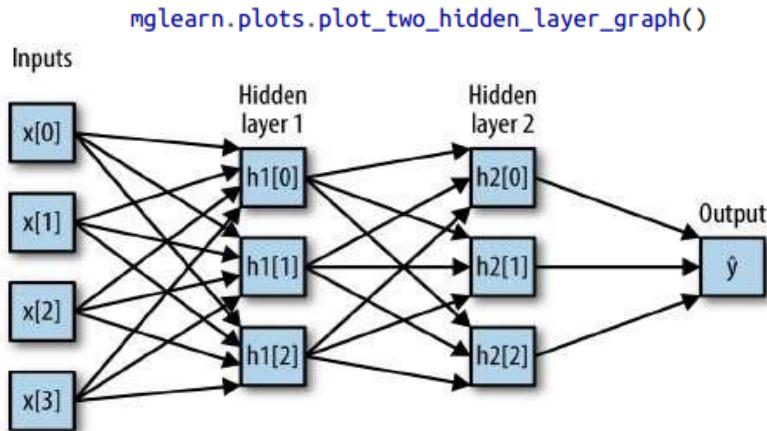
**Gambar 2.46** Fungsi aktivasi tangen hiperbolik dan fungsi aktivasi linier yang diperbaiki

Untuk jaringan saraf kecil yang digambarkan pada Gambar 2.45 rumus lengkap untuk menghitung  $\hat{y}$  dalam kasus regresi adalah (bila menggunakan tanh nonlinier):

$$\begin{aligned} h[0] &= \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3]) \\ h[1] &= \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3]) \\ h[2] &= \tanh(w[0, 0] * x[0] + w[1, 0] * x[1] + w[2, 0] * x[2] + w[3, 0] * x[3]) \\ \hat{y} &= v[0] * h[0] + v[1] * h[1] + v[2] * h[2] \end{aligned}$$

Di sini,  $w$  adalah bobot antara input  $x$  dan lapisan tersembunyi  $h$ , dan  $v$  adalah bobot antara lapisan tersembunyi  $h$  dan output  $\hat{y}$ . Bobot  $v$  dan  $w$  dipelajari dari data,  $x$  adalah fitur input, adalah output yang dihitung, dan  $h$  adalah komputasi perantara. Parameter penting yang perlu diatur oleh pengguna adalah jumlah node pada lapisan tersembunyi. Ini bisa sekecil 10 untuk kumpulan data yang sangat kecil atau sederhana dan sebesar 10.000 untuk data yang sangat kompleks. Hal ini juga memungkinkan untuk menambahkan lapisan tersembunyi tambahan, seperti yang ditunjukkan pada Gambar 2.47:

In[92]:



**Gambar 2.47** Perceptron multilayer dengan dua lapisan tersembunyi

Memiliki jaringan saraf besar yang terdiri dari banyak lapisan komputasi inilah yang mengilhami istilah “pembelajaran mendalam.”

### Menyetel jaringan saraf

Mari kita lihat cara kerja MLP dengan menerapkan MLP Classifier ke dataset `two_moons` yang kita gunakan sebelumnya dalam bab ini. Hasilnya ditunjukkan pada Gambar 2.48:

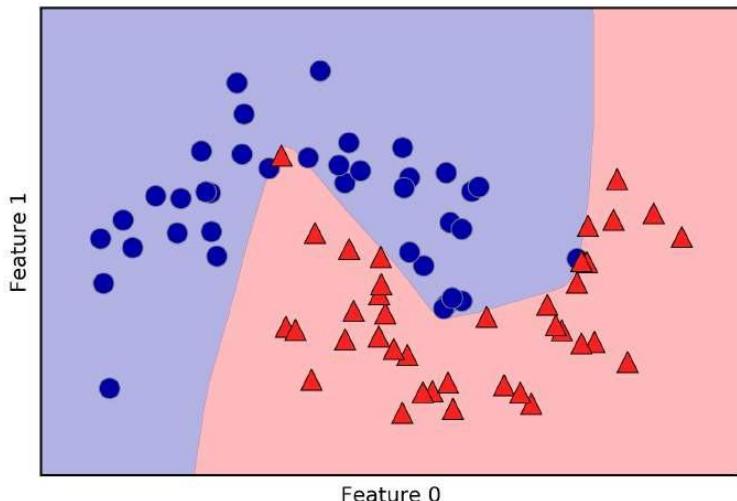
In[93]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
                                                    random_state=42)

mlp = MLPClassifier(algorithm='l-bfgs', random_state=0).fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



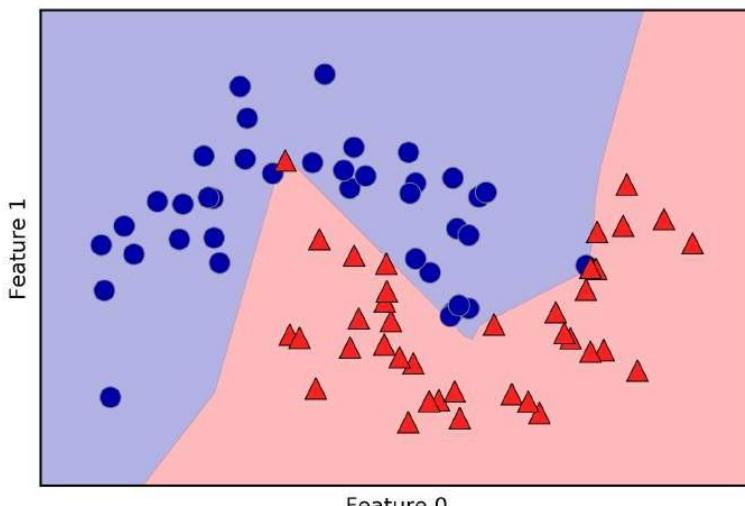
**Gambar 2.48** Batas keputusan dipelajari oleh jaringan saraf dengan 100 unit tersembunyi pada set data `two_moons`

Seperti yang Anda lihat, jaringan saraf mempelajari batas keputusan yang sangat nonlinier tetapi relatif mulus. Kami menggunakan algoritma='l-bfgs', yang akan kita bahas nanti.

Secara default, MLP menggunakan 100 node tersembunyi, yang cukup banyak untuk dataset kecil ini. Kita dapat mengurangi jumlah (yang mengurangi kompleksitas model) dan tetap mendapatkan hasil yang baik (Gambar 2.49):

In[94]:

```
mlp = MLPClassifier(algorithm='l-bfgs', random_state=0, hidden_layer_sizes=[10])
mlp.fit(X_train, y_train)
mlearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mlearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 2.49** Batas keputusan dipelajari oleh jaringan saraf dengan 10 unit tersembunyi pada dataset `two_moons`

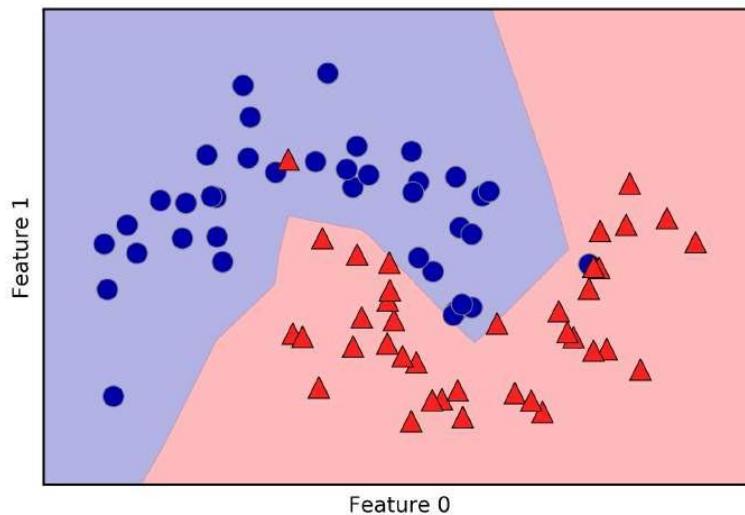
Dengan hanya 10 unit tersembunyi, batas keputusan terlihat lebih kasar. Nonlinier default adalah relu, ditunjukkan pada Gambar 2.46. Dengan satu lapisan tersembunyi, ini berarti fungsi keputusan akan terdiri dari 10 segmen garis lurus. Jika kita menginginkan batas keputusan yang lebih halus, kita dapat menambahkan lebih banyak unit tersembunyi (seperti pada Gambar 2.49), menambahkan lapisan tersembunyi kedua (Gambar 2.50), atau menggunakan tanh nonlinier (Gambar 2.51):

**In[95]:**

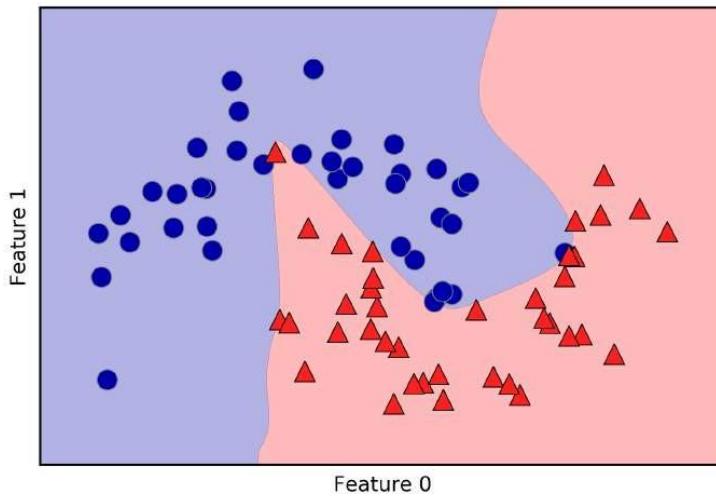
```
# using two hidden layers, with 10 units each
mlp = MLPClassifier(algorithm='l-bfgs', random_state=0,
                     hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

**In[96]:**

```
# using two hidden layers, with 10 units each, now with tanh nonlinearity
mlp = MLPClassifier(algorithm='l-bfgs', activation='tanh',
                     random_state=0, hidden_layer_sizes=[10, 10])
mlp.fit(X_train, y_train)
mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3)
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 2.50** Batas keputusan dipelajari menggunakan 2 lapisan tersembunyi dengan masing-masing 10 unit tersembunyi, dengan fungsi aktivasi langsung

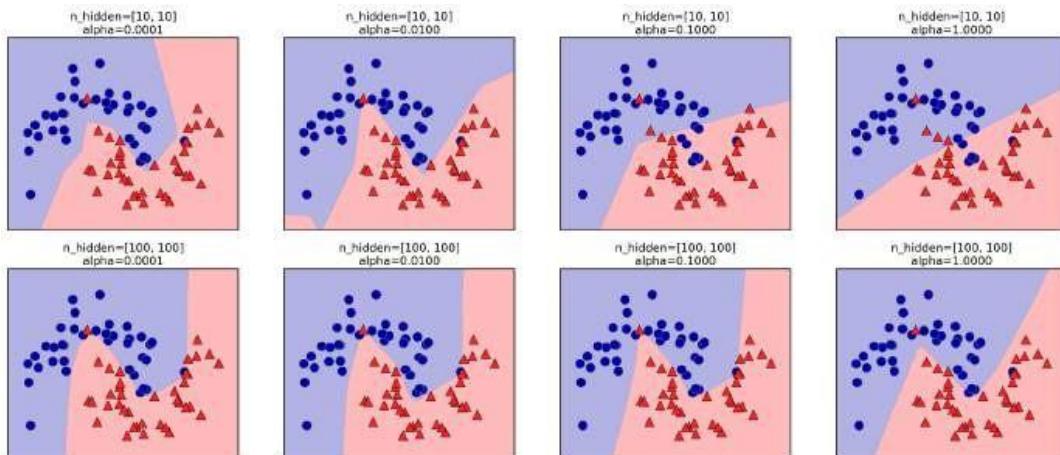


**Gambar 2.51** Batas keputusan dipelajari menggunakan 2 lapisan tersembunyi dengan masing-masing 10 unit tersembunyi, dengan fungsi aktivasi tanah

Akhirnya, kami juga dapat mengontrol kompleksitas jaringan saraf dengan menggunakan penalti L2 untuk mengecilkan bobot menuju nol, seperti yang kami lakukan dalam regresi ridge dan pengklasifikasi linier. Parameter untuk ini di MLPClassifier adalah alfa (seperti dalam model regresi linier), dan secara default disetel ke nilai yang sangat rendah (sedikit regularisasi). Gambar 2.52 menunjukkan pengaruh nilai alfa yang berbeda pada dataset two\_moons, menggunakan dua lapisan tersembunyi masing-masing 10 atau 100 unit:

In[97]:

```
fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for axx, n_hidden_nodes in zip(axes, [10, 100]):
    for ax, alpha in zip(axx, [0.0001, 0.01, 0.1, 1]):
        mlp = MLPClassifier(algorithm='l-bfgs', random_state=0,
                            hidden_layer_sizes=[n_hidden_nodes, n_hidden_nodes],
                            alpha=alpha)
        mlp.fit(X_train, y_train)
        mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3, ax=ax)
        mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train, ax=ax)
        ax.set_title("n_hidden=[{}, {}]\nalpha={:.4f}".format(
            n_hidden_nodes, n_hidden_nodes, alpha))
```



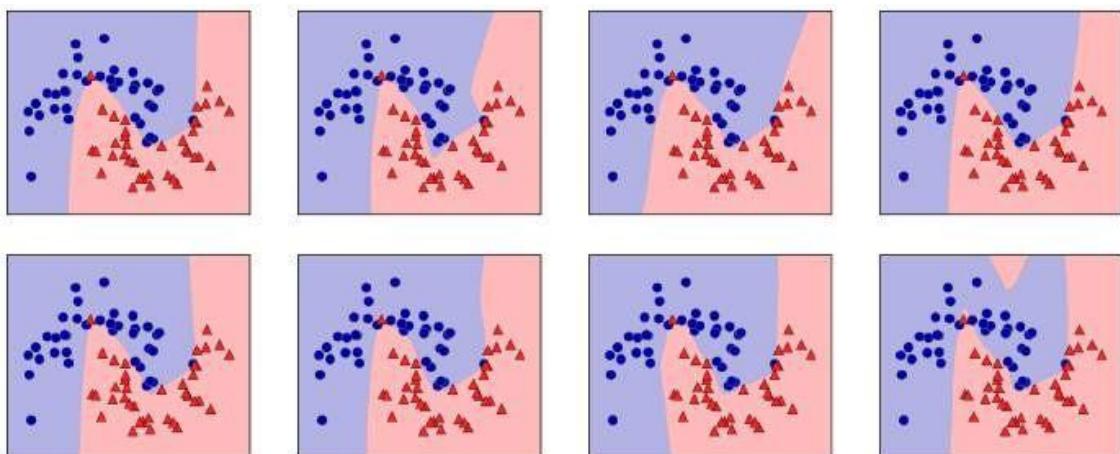
**Gambar 2.52** Fungsi keputusan untuk jumlah unit tersembunyi yang berbeda dan pengaturan parameter alfa yang berbeda

Seperti yang mungkin telah Anda sadari sekarang, ada banyak cara untuk mengontrol kompleksitas jaringan saraf: jumlah lapisan tersembunyi, jumlah unit di setiap lapisan tersembunyi, dan regularisasi ( $\alpha$ ). Sebenarnya ada lebih banyak lagi, yang tidak akan kita bahas di sini.

Properti penting dari jaringan saraf adalah bahwa bobotnya ditetapkan secara acak sebelum pembelajaran dimulai, dan inisialisasi acak ini memengaruhi model yang dipelajari. Itu berarti bahwa meskipun menggunakan parameter yang sama persis, kita dapat memperoleh model yang sangat berbeda ketika menggunakan benih acak yang berbeda. Jika jaringannya besar, dan kompleksitasnya dipilih dengan benar, ini seharusnya tidak terlalu memengaruhi akurasi, tetapi perlu diingat (terutama untuk jaringan yang lebih kecil). Gambar 2.53 menunjukkan plot beberapa model, semuanya dipelajari dengan pengaturan parameter yang sama:

In[98]:

```
fig, axes = plt.subplots(2, 4, figsize=(20, 8))
for i, ax in enumerate(axes.ravel()):
    mlp = MLPClassifier(algorithm='l-bfgs', random_state=i,
                         hidden_layer_sizes=[100, 100])
    mlp.fit(X_train, y_train)
    mglearn.plots.plot_2d_separator(mlp, X_train, fill=True, alpha=.3, ax=ax)
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train, ax=ax)
```



**Gambar 2.53** Fungsi keputusan dipelajari dengan parameter yang sama tetapi inisialisasi acak yang berbeda

Untuk mendapatkan pemahaman yang lebih baik tentang jaringan saraf pada dunia nyata, mari terapkan `MLPClassifier` ke kumpulan data Kanker Payudara. Kami mulai dengan parameter default:

**In[99]:**

```
print("Cancer data per-feature maxima:\n{}".format(cancer.data.max(axis=0)))
```

**Out[99]:**

Cancer data per-feature maxima:							
28.110	39.280	188.500	2501.000	0.163	0.345	0.427	
0.201	0.304	0.097	2.873	4.885	21.980	542.200	
0.031	0.135	0.396	0.053	0.079	0.030	36.040	
49.540	251.200	4254.000	0.223	1.058	1.252	0.291	
0.664	0.207]						

**In[100]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

mlp = MLPClassifier(random_state=42)
mlp.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(mlp.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(mlp.score(X_test, y_test)))
```

**Out[100]:**

```
Accuracy on training set: 0.92
Accuracy on test set: 0.90
```

Keakuratan MLP cukup baik, tetapi tidak sebagus model lainnya. Seperti pada contoh SVC sebelumnya, ini mungkin karena penskalaan data. Jaringan saraf juga mengharapkan semua fitur input bervariasi dengan cara yang sama, dan idealnya memiliki rata-rata 0, dan varians 1. Kita harus mengubah skala data kita sehingga memenuhi persyaratan ini. Sekali lagi, kami akan melakukannya dengan tangan di sini, tetapi kami akan memperkenalkan StandardScaler untuk melakukan ini secara otomatis di Bab 3:

**In[101]:**

```
# compute the mean value per feature on the training set
mean_on_train = X_train.mean(axis=0)
# compute the standard deviation of each feature on the training set
std_on_train = X_train.std(axis=0)

# subtract the mean, and scale by inverse standard deviation
# afterward, mean=0 and std=1
X_train_scaled = (X_train - mean_on_train) / std_on_train
# use THE SAME transformation (using training mean and std) on the test set
X_test_scaled = (X_test - mean_on_train) / std_on_train

mlp = MLPClassifier(random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

**Out[101]:**

```
Accuracy on training set: 0.991
Accuracy on test set: 0.965

ConvergenceWarning:
  Stochastic Optimizer: Maximum iterations reached and the optimization
  hasn't converged yet.
```

Hasilnya jauh lebih baik setelah penskalaan, dan sudah cukup kompetitif. Kami mendapat peringatan dari model, yang memberi tahu kami bahwa jumlah maksimum iterasi telah tercapai. Ini adalah bagian dari algoritma adam untuk mempelajari model, dan memberitahu kita bahwa kita harus meningkatkan jumlah iterasi:

**In[102]:**

```
mlp = MLPClassifier(max_iter=1000, random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

**Out[102]:**

```
Accuracy on training set: 0.995
Accuracy on test set: 0.965
```

Meningkatkan jumlah iterasi hanya meningkatkan kinerja set pelatihan, bukan kinerja generalisasi. Meski begitu, model ini berkinerja cukup baik. Karena ada beberapa kesenjangan antara pelatihan dan kinerja pengujian, kami mungkin mencoba mengurangi kompleksitas model untuk mendapatkan kinerja generalisasi yang lebih baik. Di sini, kami memilih untuk meningkatkan parameter alfa (cukup agresif, dari 0,0001 menjadi 1) untuk menambahkan regularisasi bobot yang lebih kuat:

**In[103]:**

```
mlp = MLPClassifier(max_iter=1000, alpha=1, random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    mlp.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(mlp.score(X_test_scaled, y_test)))
```

**Out[103]:**

```
Accuracy on training set: 0.988
Accuracy on test set: 0.972
```

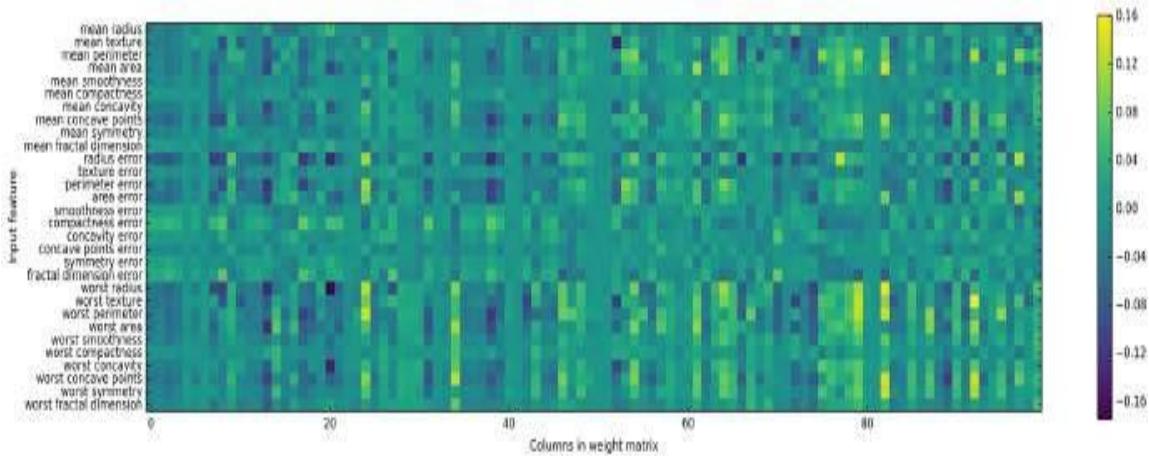
Ini mengarah pada kinerja yang setara dengan model terbaik sejauh ini.<sup>12</sup>

Meskipun dimungkinkan untuk menganalisis apa yang telah dipelajari jaringan saraf, ini biasanya jauh lebih sulit daripada menganalisis model linier atau model berbasis pohon. Salah satu cara untuk mengintrospeksi apa yang dipelajari adalah dengan melihat bobot dalam model. Anda dapat melihat contohnya di galeri contoh scikit-learn. Untuk dataset Kanker Payudara, ini mungkin agak sulit dipahami. Plot berikut (Gambar 2.54) menunjukkan bobot yang dipelajari menghubungkan input ke lapisan tersembunyi pertama. Baris dalam plot ini

sesuai dengan 30 fitur input, sedangkan kolom sesuai dengan 100 unit tersembunyi. Warna terang mewakili nilai positif yang besar, sedangkan warna gelap mewakili nilai negatif:

In[104]:

```
plt.figure(figsize=(20, 5))
plt.imshow(mlp.coefs_[0], interpolation='none', cmap='viridis')
plt.yticks(range(30), cancer.feature_names)
plt.xlabel("Columns in weight matrix")
plt.ylabel("Input feature")
plt.colorbar()
```



**Gambar 2.54** Peta panas dari bobot lapisan pertama dalam jaringan saraf yang dipelajari pada dataset Kanker Payudara

Satu kesimpulan yang mungkin dapat kita buat adalah bahwa fitur yang memiliki bobot sangat kecil untuk semua unit tersembunyi adalah “kurang penting” bagi model. Kita dapat melihat bahwa “kehalusan rata-rata” dan “kekompakan rata-rata”, selain fitur yang ditemukan antara “kesalahan kehalusan” dan “kesalahan dimensi fraktal”, memiliki bobot yang relatif rendah dibandingkan fitur lainnya. Ini bisa berarti bahwa ini adalah fitur yang kurang penting atau mungkin kami tidak merepresentasikannya dengan cara yang dapat digunakan oleh jaringan saraf.

Kami juga dapat memvisualisasikan bobot yang menghubungkan lapisan tersembunyi ke lapisan keluaran, tetapi itu lebih sulit untuk ditafsirkan.

Sementara MLPClassifier dan MLPRegressor menyediakan antarmuka yang mudah digunakan untuk arsitektur jaringan saraf yang paling umum, mereka hanya menangkap sebagian kecil dari apa yang mungkin dengan jaringan saraf. Jika Anda tertarik untuk bekerja dengan model yang lebih fleksibel atau lebih besar, kami mendorong Anda untuk melihat melampaui scikit-belajar ke perpustakaan pembelajaran mendalam yang fantastis yang ada di luar sana. Untuk pengguna Python, yang paling mapan adalah keras, lasagna, dan tensor-flow. Lasagna dibangun di perpustakaan theano, sementara keras dapat menggunakan tensor-flow atau theano.

Perpustakaan ini menyediakan antarmuka yang jauh lebih fleksibel untuk membangun jaringan saraf dan melacak kemajuan pesat dalam penelitian pembelajaran mendalam. Semua perpustakaan pembelajaran mendalam yang populer juga memungkinkan penggunaan unit

pemrosesan grafis (GPU) kinerja tinggi, yang tidak didukung oleh scikit-learn. Menggunakan GPU memungkinkan kami untuk mempercepat komputasi dengan faktor 10x hingga 100x, dan itu penting untuk menerapkan metode pembelajaran mendalam ke kumpulan data skala besar.

### **Kekuatan, kelemahan, dan parameter**

Jaringan saraf telah muncul kembali sebagai model canggih dalam banyak aplikasi pembelajaran mesin. Salah satu keuntungan utama mereka adalah bahwa mereka mampu menangkap informasi yang terkandung dalam sejumlah besar data dan membangun model yang sangat kompleks. Dengan waktu komputasi yang cukup, data, dan penyetelan parameter yang cermat, jaringan saraf sering mengalahkan algoritme pembelajaran mesin lainnya (untuk tugas klasifikasi dan regresi).

Ini membawa kita ke sisi negatifnya. Jaringan saraf—khususnya yang besar dan kuat—sering kali membutuhkan waktu lama untuk dilatih. Mereka juga membutuhkan pra-pemrosesan data yang cermat, seperti yang kita lihat di sini. Mirip dengan SVM, mereka bekerja paling baik dengan data "homogen", di mana semua fitur memiliki arti yang sama. Untuk data yang memiliki jenis fitur yang sangat berbeda, model berbasis pohon mungkin berfungsi lebih baik. Menyetel parameter jaringan saraf juga merupakan seni tersendiri. Dalam percobaan kami, kami hampir tidak menemukan cara yang mungkin untuk menyesuaikan model jaringan saraf dan cara melathinya.

Memperkirakan kompleksitas dalam jaringan saraf. Parameter yang paling penting adalah jumlah lapisan dan jumlah unit tersembunyi per lapisan. Anda harus mulai dengan satu atau dua lapisan tersembunyi, dan mungkin berkembang dari sana. Jumlah node per lapisan tersembunyi sering kali serupa dengan jumlah fitur input, tetapi jarang lebih tinggi daripada di bawah hingga pertengahan ribuan.

Ukuran yang membantu ketika memikirkan kompleksitas model jaringan saraf adalah jumlah bobot atau koefisien yang dipelajari. Jika Anda memiliki dataset klasifikasi biner dengan 100 fitur, dan Anda memiliki 100 unit tersembunyi, maka ada  $100 * 100 = 10.000$  bobot antara input dan lapisan tersembunyi pertama. Ada juga  $100 * 1 = 100$  bobot antara lapisan tersembunyi dan lapisan keluaran, dengan total sekitar 10.100 bobot. Jika Anda menambahkan lapisan tersembunyi kedua dengan 100 unit tersembunyi, akan ada  $100 * 100 = 10.000$  bobot lagi dari lapisan tersembunyi pertama ke lapisan tersembunyi kedua, sehingga total bobot menjadi 20.100. Jika sebaliknya Anda menggunakan satu lapisan dengan 1.000 unit tersembunyi, Anda mempelajari  $100 * 1.000 = 100.000$  bobot dari input ke lapisan tersembunyi dan  $1.000 * 1$  bobot dari lapisan tersembunyi ke lapisan output, dengan total 101.000. Jika Anda menambahkan lapisan tersembunyi kedua, Anda menambahkan  $1.000 * 1.000 = 1.000.000$  bobot, dengan total kekalahan 1.101.000—50 kali lebih besar dari model dengan dua lapisan tersembunyi berukuran 100.

Cara umum untuk menyesuaikan parameter dalam jaringan saraf adalah pertama-tama membuat jaringan yang cukup besar untuk overfit, memastikan bahwa tugas tersebut benar-benar dapat dipelajari oleh jaringan. Kemudian, setelah Anda mengetahui data pelatihan dapat dipelajari, baik mengecilkan jaringan atau meningkatkan alfa untuk menambahkan regularisasi, yang akan meningkatkan kinerja generalisasi.

Dalam percobaan kami, kami sebagian besar berfokus pada definisi model: jumlah lapisan dan node per lapisan, regularisasi, dan nonlinier. Ini menentukan model yang ingin kita pelajari. Ada juga pertanyaan tentang bagaimana mempelajari model, atau algoritma yang digunakan untuk mempelajari parameter, yang diatur menggunakan parameter algoritma. Ada dua pilihan algoritma yang mudah digunakan.

Standarnya adalah 'adam', yang berfungsi dengan baik di sebagian besar situasi tetapi cukup sensitif terhadap penskalaan data (jadi penting untuk selalu menskalakan data Anda ke 0 mean dan varians unit). Yang lainnya adalah 'l-bfgs', yang cukup kuat tetapi mungkin memakan waktu lama pada model yang lebih besar atau kumpulan data yang lebih besar. Ada juga opsi 'sgd' yang lebih canggih, yang digunakan oleh banyak peneliti pembelajaran mendalam. Opsi 'sgd' hadir dengan banyak parameter tambahan yang perlu disesuaikan untuk hasil terbaik. Anda dapat menemukan semua parameter ini dan definisinya di panduan pengguna. Saat mulai bekerja dengan MLP, kami sarankan untuk tetap menggunakan 'adam' dan 'l-bfgs'.



### Cocok Mengatur Ulang Model

Properti penting dari model scikit-learn adalah bahwa memanggil fit akan selalu mengatur ulang semua model yang dipelajari sebelumnya. Jadi, jika Anda membuat model pada satu kumpulan data, lalu memanggil fit lagi pada kumpulan data yang berbeda, model akan "melupakan" semua yang dipelajarinya dari kumpulan data pertama. Anda dapat memanggil fit sesering yang Anda suka pada model, dan hasilnya akan sama dengan panggilan fit pada model "baru".

## 2.12 PERKIRAAN KETIDAKPASTIAN DARI PENGKLASIFIKASI

Bagian lain yang berguna dari antarmuka scikit-learn yang belum kita bicarakan adalah kemampuan pengklasifikasi untuk memberikan perkiraan prediksi yang tidak pasti. Seringkali, Anda tidak hanya tertarik pada kelas mana yang diprediksi oleh pengklasifikasi untuk titik tes tertentu, tetapi juga seberapa yakin bahwa ini adalah kelas yang tepat. Dalam praktiknya, berbagai jenis kesalahan menyebabkan hasil yang sangat berbeda dalam aplikasi dunia nyata. Bayangkan sebuah pengujian aplikasi medis untuk kanker. Membuat prediksi positif palsu dapat menyebabkan pasien menjalani tes tambahan, sementara prediksi negatif palsu dapat menyebabkan penyakit serius yang tidak diobati. Kami akan membahas topik ini secara lebih rinci di Bab 6.

Ada dua fungsi berbeda dalam scikit-learn yang dapat digunakan untuk mendapatkan estimasi ketidakpastian dari pengklasifikasi: fungsi\_keputusan dan prediksi\_proba. Sebagian besar (tetapi tidak semua) pengklasifikasi memiliki setidaknya satu dari mereka, dan banyak pengklasifikasi memiliki keduanya. Mari kita lihat apa yang dilakukan kedua fungsi ini pada dataset dua dimensi sintetik, saat membangun pengklasifikasi GradientBoostingClassifier, yang memiliki metode decision\_function dan metode predict\_proba:

**In[105]:**

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_blobs, make_circles
X, y = make_circles(noise=0.25, factor=0.5, random_state=1)

# we rename the classes "blue" and "red" for illustration purposes
y_named = np.array(["blue", "red"])[y]

# we can call train_test_split with arbitrarily many arrays;
# all will be split in a consistent manner
X_train, X_test, y_train_named, y_test_named, y_train, y_test = \
    train_test_split(X, y_named, y, random_state=0)

# build the gradient boosting model
gbdt = GradientBoostingClassifier(random_state=0)
gbdt.fit(X_train, y_train_named)
```

## 2.13 FUNGSI KEPUTUSAN

Dalam kasus klasifikasi biner, nilai kembalian dari fungsi\_keputusan berbentuk (n\_sampel,), dan mengembalikan satu angka titik-mengambang untuk setiap sampel:

**In[106]:**

```
print("X_test.shape: {}".format(X_test.shape))
print("Decision function shape: {}".format(
    gbdt.decision_function(X_test).shape))
```

**Out[106]:**

```
X_test.shape: (25, 2)
Decision function shape: (25,)
```

Nilai ini mengkodekan seberapa kuat model percaya titik data milik kelas "positif", dalam hal ini kelas 1. Nilai positif menunjukkan preferensi untuk kelas positif, dan nilai negatif menunjukkan preferensi untuk "negatif" (lainnya) kelas:

**In[107]:**

```
# show the first few entries of decision_function
print("Decision function:\n{}".format(gbdt.decision_function(X_test)[:6]))
```

**Out[107]:**

```
Decision function:
[ 4.136 -1.683 -3.951 -3.626  4.29   3.662]
```

Kita dapat memulihkan prediksi dengan hanya melihat tanda dari fungsi keputusan:

**In[108]:**

```
print("Thresholded decision function:\n{}".format(
    gbdt.decision_function(X_test) > 0))
print("Predictions:\n{}".format(gbdt.predict(X_test)))
```

**Out[108]:**

```
Thresholded decision function:
[ True False False False  True  True False  True  True  True False  True
  True False  True False False False  True  True  True  True False
  False]
Predictions:
['red' 'blue' 'blue' 'blue' 'red' 'red' 'blue' 'red' 'red' 'red' 'blue'
 'red' 'red' 'blue' 'red' 'blue' 'blue' 'blue' 'red' 'red' 'red' 'red'
 'red' 'blue' 'blue']
```

Untuk klasifikasi biner, kelas “negatif” selalu merupakan entri pertama dari atribut `class_`, dan kelas “positif” adalah entri kedua dari `class_`. Jadi jika Anda ingin sepenuhnya memulihkan output dari prediksi, Anda perlu menggunakan atribut `class_`:

**In[109]:**

```
# make the boolean True/False into 0 and 1
greater_zero = (gbrt.decision_function(X_test) > 0).astype(int)
# use 0 and 1 as indices into classes_
pred = gbrt.classes_[greater_zero]
# pred is the same as the output of gbrt.predict
print("pred is equal to predictions: {}".format(
    np.all(pred == gbrt.predict(X_test))))
```

**Out[109]:**

```
pred is equal to predictions: True
```

Rentang dari `decision_function` dapat berubah-ubah, dan bergantung pada data dan parameter model:

**In[110]:**

```
decision_function = gbrt.decision_function(X_test)
print("Decision function minimum: {:.2f} maximum: {:.2f}".format(
    np.min(decision_function), np.max(decision_function)))
```

**Out[110]:**

```
Decision function minimum: -7.69 maximum: 4.29
```

Penskalaan arbitrer ini membuat output dari `decision_function` sering kali sulit untuk diinterpretasikan.

Dalam contoh berikut, kita memplot fungsi\_keputusan untuk semua titik dalam bidang 2D menggunakan kode warna, di samping visualisasi batas keputusan, seperti yang kita lihat sebelumnya. Kami menunjukkan titik pelatihan sebagai lingkaran dan data uji sebagai segitiga (Gambar 2.55):

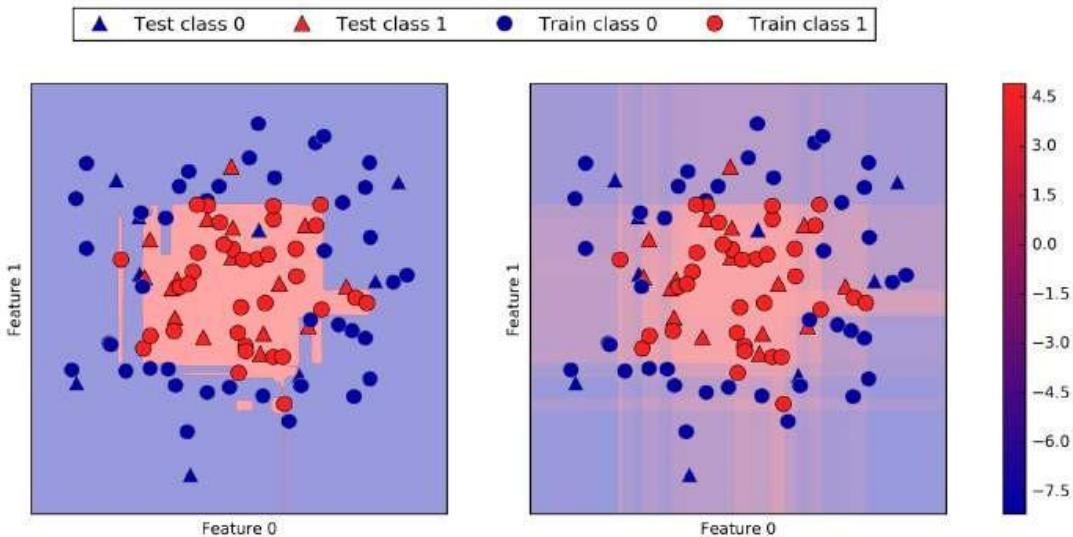
**In[111]:**

```
fig, axes = plt.subplots(1, 2, figsize=(13, 5))
mglearn.tools.plot_2d_separator(gbrt, X, ax=axes[0], alpha=.4,
                                fill=True, cm=mglearn.cm2)
scores_image = mglearn.tools.plot_2d_scores(gbrt, X, ax=axes[1],
                                            alpha=.4, cm=mglearn.ReBl)
```

```

for ax in axes:
    # plot training and test points
    mglearn.discrete_scatter(X_test[:, 0], X_test[:, 1], y_test,
                             markers='^', ax=ax)
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train,
                             markers='o', ax=ax)
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
cbar = plt.colorbar(scores_image, ax=axes.tolist())
axes[0].legend(["Test class 0", "Test class 1", "Train class 0",
                "Train class 1"], ncol=4, loc=(.1, 1.1))

```



**Gambar 2.55** Batas keputusan (kiri) dan fungsi keputusan (kanan) untuk model peningkatan gradien pada kumpulan data mainan dua dimensi

Pengkodean tidak hanya hasil yang diprediksi tetapi juga seberapa pasti pengklasifikasi memberikan informasi tambahan. Namun, dalam visualisasi ini, sulit untuk melihat batas antara dua kelas.

## 2.14 MEMPREDIKSI PROBABILITAS

Keluaran dari `predict_proba` adalah probabilitas untuk setiap kelas, dan seringkali lebih mudah dipahami daripada keluaran dari `fungsi_keputusan`. Itu selalu berbentuk (`n_samples`, 2) untuk klasifikasi biner:

**In[112]:**

```
print("Shape of probabilities: {}".format(gbdt.predict_proba(X_test).shape))
```

**Out[112]:**

```
Shape of probabilities: (25, 2)
```

Entri pertama di setiap baris adalah estimasi probabilitas dari kelas pertama, dan entri kedua adalah estimasi probabilitas dari kelas kedua. Karena probabilitas, output dari `predict_proba` selalu antara 0 dan 1, dan jumlah entri untuk kedua kelas selalu 1:

In[113]:

```
# show the first few entries of predict_proba
print("Predicted probabilities:\n{}".format(
    gbrt.predict_proba(X_test[:6])))
```

Out[113]:

```
Predicted probabilities:
[[ 0.016  0.984]
 [ 0.843  0.157]
 [ 0.981  0.019]
 [ 0.974  0.026]
 [ 0.014  0.986]
 [ 0.025  0.975]]
```

Karena peluang kedua kelas berjumlah 1, tepat salah satu kelas akan berada di atas kepastian 50%. Kelas itu adalah salah satu yang diprediksi. Anda dapat melihat pada output sebelumnya bahwa classifier relatif pasti untuk sebagian besar poin. Seberapa baik ketidakpastian sebenarnya mencerminkan ketidakpastian dalam data tergantung pada model dan parameteranya. Model yang lebih overfitted cenderung membuat prediksi yang lebih pasti, meskipun mungkin salah. Model dengan kompleksitas yang lebih kecil biasanya memiliki lebih banyak ketidakpastian dalam prediksinya. Sebuah model disebut dikalibrasi jika ketidakpastian yang dilaporkan benar-benar cocok dengan seberapa benarnya—dalam model yang dikalibrasi, prediksi yang dibuat dengan kepastian 70% akan benar 70% dari waktu.

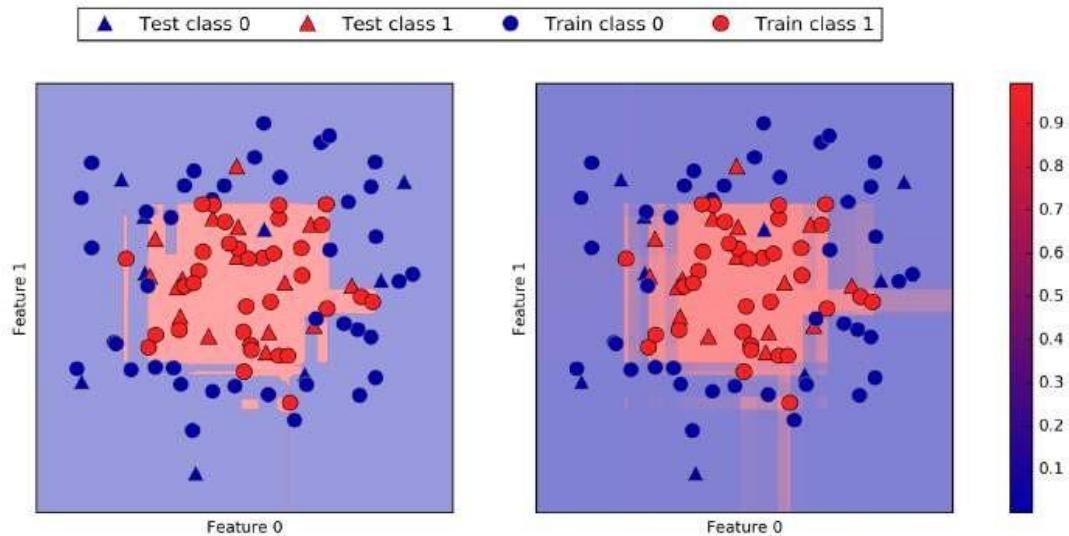
Dalam contoh berikut (Gambar 2.56) kami kembali menunjukkan batas keputusan pada dataset, di sebelah probabilitas kelas untuk kelas 1:

In[114]:

```
fig, axes = plt.subplots(1, 2, figsize=(13, 5))

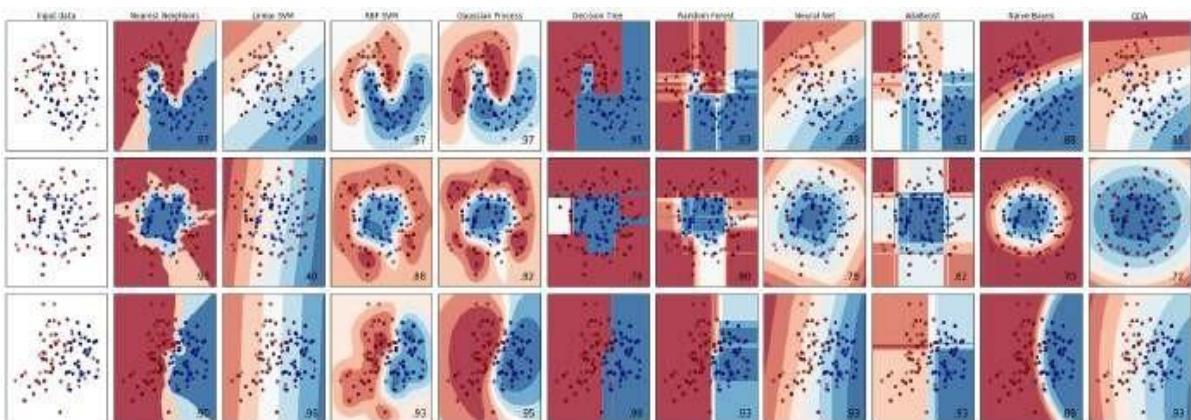
mglearn.tools.plot_2d_separator(
    gbrt, X, ax=axes[0], alpha=.4, fill=True, cm=mglearn.cm2)
scores_image = mglearn.tools.plot_2d_scores(
    gbrt, X, ax=axes[1], alpha=.5, cm=mglearn.ReBl, function='predict_proba')

for ax in axes:
    # plot training and test points
    mglearn.discrete_scatter(X_test[:, 0], X_test[:, 1], y_test,
                             markers='^', ax=ax)
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train,
                             markers='o', ax=ax)
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
cbar = plt.colorbar(scores_image, ax=axes.tolist())
axes[0].legend(["Test class 0", "Test class 1", "Train class 0",
                "Train class 1"], ncol=4, loc=(.1, 1.1))
```



**Gambar 2.56** Batas keputusan (kiri) dan probabilitas yang diprediksi untuk model peningkatan gradien ditunjukkan pada *Gambar 2.55*

Batas-batas di plot ini jauh lebih jelas, dan area kecil ketidakpastian terlihat jelas. Situs web scikit-learn memiliki perbandingan banyak model dan seperti apa perkiraan ketidakpastiannya. Kami telah mereproduksi ini pada Gambar 2.57, dan kami mendorong Anda untuk pergi melalui contoh di sana.



**Gambar 2.57** Perbandingan beberapa pengklasifikasi dalam scikit-learn pada kumpulan data sintetis (gambar milik <http://scikit-learn.org>)

## 2.15 KETIDAKPASTIAN DALAM KLASIFIKASI MULTIKLASIFIKASI

Sejauh ini, kita hanya berbicara tentang estimasi ketidakpastian dalam klasifikasi biner. Tetapi metode `decision_function` dan `predict_proba` juga berfungsi dalam pengaturan multikelas. Mari kita terapkan pada dataset Iris, yang merupakan dataset klasifikasi tiga kelas:

**In[115]:**

```
from sklearn.datasets import load_iris

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=42)

gbdt = GradientBoostingClassifier(learning_rate=0.01, random_state=0)
gbdt.fit(X_train, y_train)
```

**In[116]:**

```
print("Decision function shape: {}".format(gbdt.decision_function(X_test).shape))
# plot the first few entries of the decision function
print("Decision function:\n{}".format(gbdt.decision_function(X_test)[:6, :]))
```

**Out[116]:**

```
Decision function shape: (38, 3)
Decision function:
[[ -0.529  1.466 -0.504]
 [ 1.512 -0.496 -0.503]
 [-0.524 -0.468  1.52 ]
 [-0.529  1.466 -0.504]
 [-0.531  1.282  0.215]
 [ 1.512 -0.496 -0.503]]
```

Dalam kasus multikelas, fungsi\_keputusan memiliki bentuk (n\_samples, n\_classes) dan setiap kolom memberikan "skor kepastian" untuk setiap kelas, di mana skor besar berarti kelas lebih mungkin dan skor kecil berarti kelas lebih kecil kemungkinannya. Anda dapat memulihkan prediksi dari skor ini dengan menemukan entri maksimum untuk setiap titik data:

**In[117]:**

```
print("Argmax of decision function:\n{}".format(
    np.argmax(gbdt.decision_function(X_test), axis=1)))
print("Predictions:\n{}".format(gbdt.predict(X_test)))
```

**Out[117]:**

```
Argmax of decision function:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
Predictions:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
```

Output dari predict\_proba memiliki bentuk yang sama, (n\_samples, n\_classes). Sekali lagi, probabilitas untuk kelas yang mungkin untuk setiap titik data berjumlah 1:

**In[118]:**

```
# show the first few entries of predict_proba
print("Predicted probabilities:\n{}".format(gbdt.predict_proba(X_test)[:6]))
# show that sums across rows are one
print("Sums: {}".format(gbdt.predict_proba(X_test)[:6].sum(axis=1)))
```

**Out[118]:**

```
Predicted probabilities:
[[ 0.107  0.784  0.109]
 [ 0.789  0.106  0.105]
 [ 0.102  0.108  0.789]
 [ 0.107  0.784  0.109]
 [ 0.108  0.663  0.228]
 [ 0.789  0.106  0.105]]
Sums: [ 1.  1.  1.  1.  1.  1.]
```

Kami kembali dapat memulihkan prediksi dengan menghitung argmax dari predict\_proba:

**In[119]:**

```
print("Argmax of predicted probabilities:\n{}".format(
    np.argmax(gbrt.predict_proba(X_test), axis=1)))
print("Predictions:\n{}".format(gbrt.predict(X_test)))
```

**Out[119]:**

```
Argmax of predicted probabilities:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
Predictions:
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
```

Untuk meringkas, predict\_proba dan fungsi\_keputusan selalu memiliki bentuk (n\_sampel, n\_kelas)—terlepas dari fungsi\_keputusan dalam kasus biner khusus. Dalam kasus biner, decision\_function hanya memiliki satu kolom, sesuai dengan kelas kelas “positif”[1]. Ini sebagian besar karena alasan historis.

Anda dapat memulihkan prediksi ketika ada n\_kelas banyak kolom dengan menghitung argmax di seluruh kolom. Namun berhati-hatilah, jika kelas Anda adalah string, atau Anda menggunakan bilangan bulat tetapi tidak berurutan dan dimulai dari 0. Jika Anda ingin membandingkan hasil yang diperoleh dengan prediksi dengan hasil yang diperoleh melalui decision\_function atau predict\_proba, pastikan untuk menggunakan atribut class\_ dari classifier untuk mendapatkan nama kelas yang sebenarnya:

**In[120]:**

```
logreg = LogisticRegression()

# represent each target by its class name in the iris dataset
named_target = iris.target_names[y_train]
logreg.fit(X_train, named_target)
print("unique classes in training data: {}".format(logreg.classes_))
print("predictions: {}".format(logreg.predict(X_test)[:10]))
argmax_dec_func = np.argmax(logreg.decision_function(X_test), axis=1)
print("argmax of decision function: {}".format(argmax_dec_func[:10]))
print("argmax combined with classes_: {}".format(
    logreg.classes_[argmax_dec_func][:10]))
```

**Out[120]:**

```
unique classes in training data: ['setosa' 'versicolor' 'virginica']
predictions: ['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor'
 'setosa' 'versicolor' 'virginica' 'versicolor' 'versicolor']
argmax of decision function: [1 0 2 1 1 0 1 2 1 1]
argmax combined with classes_: ['versicolor' 'setosa' 'virginica' 'versicolor'
 'versicolor' 'setosa' 'versicolor' 'virginica' 'versicolor' 'versicolor']
```

## 2.16 RINGKASAN DAN PANDANGAN

Kami memulai bab ini dengan diskusi tentang kompleksitas model, kemudian membahas generalisasi, atau mempelajari model yang mampu berkinerja baik pada data baru yang sebelumnya tidak terlihat. Ini membawa kami ke konsep underfitting, yang menggambarkan model yang tidak dapat menangkap variasi yang ada dalam data pelatihan, dan overfitting, yang menggambarkan model yang terlalu fokus pada data pelatihan dan tidak dapat menggeneralisasi ke data baru dengan baik.

Kami kemudian membahas beragam model pembelajaran mesin untuk klasifikasi dan regresi, apa kelebihan dan kekurangannya, dan bagaimana mengontrol kompleksitas model untuk masing-masing model. Kami melihat bahwa untuk banyak algoritme, pengaturan parameter yang tepat penting untuk kinerja yang baik. Beberapa algoritme juga sensitif terhadap cara kami merepresentasikan data input, dan khususnya bagaimana fitur diskalakan. Oleh karena itu, menerapkan algoritma secara membabi buta ke kumpulan data tanpa memahami asumsi yang dibuat model dan arti dari pengaturan parameter jarang akan menghasilkan model yang akurat.

Bab ini berisi banyak informasi tentang algoritme, dan Anda tidak perlu mengingat semua detail ini untuk bab-bab berikutnya. Namun, beberapa pengetahuan tentang model yang dijelaskan di sini—and yang akan digunakan dalam situasi tertentu—penting untuk berhasil menerapkan pembelajaran mesin dalam praktik. Berikut adalah ringkasan singkat tentang kapan harus menggunakan setiap model:

#### *Tetangga terdekat*

Untuk kumpulan data kecil, bagus sebagai dasar, mudah dijelaskan.

#### *Model linier*

Masuk sebagai algoritme pertama yang dicoba, bagus untuk kumpulan data yang sangat besar, bagus untuk data berdimensi sangat tinggi.

#### *Naif Bayes*

Hanya untuk klasifikasi. Bahkan lebih cepat daripada model linier, bagus untuk kumpulan data yang sangat besar dan data berdimensi tinggi. Seringkali kurang akurat dibandingkan model linier.

#### *Pohon keputusan*

Sangat cepat, tidak perlu penskalaan data, dapat divisualisasikan dan mudah dijelaskan.

#### *Hutan acak*

Hampir selalu berkinerja lebih baik daripada pohon keputusan tunggal, sangat kuat dan kuat. Tidak perlu penskalaan data. Tidak baik untuk data sparse berdimensi sangat tinggi.

#### *Pohon keputusan yang didorong gradien*

Seringkali sedikit lebih akurat daripada hutan acak. Lebih lambat untuk dilatih tetapi lebih cepat untuk diprediksi daripada hutan acak, dan lebih kecil dalam memori. Perlu lebih banyak penyetelan parameter daripada hutan acak.

#### *Mendukung mesin vektor*

Kuat untuk kumpulan data berukuran sedang dari fitur dengan arti yang sama. Memerlukan penskalaan data, sensitif terhadap parameter.

#### *Jaringan saraf*

Dapat membangun model yang sangat kompleks, terutama untuk kumpulan data yang besar. Peka terhadap penskalaan data dan pilihan parameter. Model besar membutuhkan waktu lama untuk melatih.

Saat bekerja dengan kumpulan data baru, secara umum merupakan ide yang baik untuk memulai dengan model sederhana, seperti model linier atau naive Bayes atau pengklasifikasi tetangga terdekat, dan lihat seberapa jauh Anda bisa mendapatkan. Setelah memahami lebih banyak tentang data, Anda dapat mempertimbangkan untuk beralih ke algoritme yang dapat membangun model yang lebih kompleks, seperti hutan acak, pohon keputusan yang didorong gradien, SVM, atau jaringan saraf.

Anda sekarang harus berada dalam posisi di mana Anda memiliki beberapa gagasan tentang bagaimana menerapkan, menyesuaikan, dan menganalisis model yang kita diskusikan di sini. Dalam bab ini, kami fokus pada kasus klasifikasi biner, karena ini biasanya paling mudah dipahami. Namun, sebagian besar algoritma yang disajikan memiliki varian klasifikasi dan regresi, dan semua algoritma klasifikasi mendukung klasifikasi biner dan multikelas. Coba terapkan salah satu dari algoritme ini ke kumpulan data bawaan di scikit-learn, seperti boston\_housing atau kumpulan data diabetes untuk regresi, atau kumpulan data digit untuk klasifikasi multikelas. Bermain-main dengan algoritme pada kumpulan data yang berbeda akan memberi Anda perasaan yang lebih baik tentang berapa lama mereka perlu dilatih, betapa mudahnya menganalisis model, dan seberapa sensitifnya mereka terhadap representasi data.

Sementara kami menganalisis konsekuensi dari pengaturan parameter yang berbeda untuk algoritma yang kami selidiki, membangun model yang benar-benar menggeneralisasi dengan baik ke data baru dalam produksi sedikit lebih rumit dari itu. Kita akan melihat bagaimana menyesuaikan parameter dengan benar dan bagaimana menemukan parameter yang baik secara otomatis di Bab 6.

Namun, pertama-tama, kita akan menyelami lebih detail ke dalam pembelajaran dan praproses tanpa pengawasan di bab berikutnya.

## BAB 3

### PEMBELAJARAN DAN PREPROCESSING TANPA PENGAWASAN

Keluarga kedua dari algoritma pembelajaran mesin yang akan kita bahas adalah algoritma pembelajaran tanpa pengawasan. Pembelajaran tanpa pengawasan mencakup semua jenis pembelajaran mesin di mana tidak ada keluaran yang diketahui, tidak ada guru yang menginstruksikan algoritma pembelajaran. Dalam pembelajaran tanpa pengawasan, algoritma pembelajaran hanya ditampilkan data input dan diminta untuk mengekstrak pengetahuan dari data ini.

#### **3.1 JENIS PEMBELAJARAN TANPA PENGAWASAN**

Kita akan melihat dua jenis pembelajaran tanpa pengawasan dalam bab ini: transformasi kumpulan data dan pengelompokan.

Transformasi dataset tanpa pengawasan adalah algoritma yang membuat representasi baru dari data yang mungkin lebih mudah dipahami oleh manusia atau algoritma pembelajaran mesin lainnya dibandingkan dengan representasi asli dari data. Aplikasi umum dari transformasi tanpa pengawasan adalah pengurangan dimensi, yang mengambil representasi data berdimensi tinggi, yang terdiri dari banyak fitur, dan menemukan cara baru untuk merepresentasikan data ini yang merangkum karakteristik penting dengan fitur yang lebih sedikit. Aplikasi umum untuk reduksi dimensi adalah reduksi menjadi dua dimensi untuk tujuan visualisasi.

Aplikasi lain untuk transformasi tanpa pengawasan adalah menemukan bagian atau komponen yang "membentuk" data. Contohnya adalah ekstraksi topik pada kumpulan dokumen teks. Di sini, tugasnya adalah menemukan topik yang tidak diketahui yang dibicarakan di setiap dokumen, dan mempelajari topik apa yang muncul di setiap dokumen. Ini berguna untuk melacak diskusi tentang tema-tema seperti pemilu, kontrol senjata, atau bintang pop di media sosial.

Algoritma pengelompokan, di sisi lain, mempartisi data menjadi kelompok-kelompok berbeda dari item serupa. Perhatikan contoh mengunggah foto ke situs media sosial. Untuk memungkinkan Anda mengatur gambar, situs mungkin ingin mengelompokkan gambar yang memperlihatkan orang yang sama. Namun, situs tidak tahu gambar mana yang menunjukkan siapa, dan tidak tahu berapa banyak orang berbeda yang muncul di koleksi foto Anda. Pendekatan yang masuk akal adalah mengekstrak semua wajah dan membaginya menjadi kelompok wajah yang terlihat serupa. Mudah-mudahan, ini sesuai dengan orang yang sama, dan gambar dapat dikelompokkan bersama untuk Anda.

#### **Tantangan dalam Pembelajaran Tanpa Pengawasan**

Tantangan utama dalam pembelajaran tanpa pengawasan adalah mengevaluasi apakah algoritma mempelajari sesuatu yang berguna. Algoritme pembelajaran tanpa pengawasan biasanya diterapkan pada data yang tidak berisi informasi label apa pun, jadi kami tidak tahu seperti apa keluaran yang tepat. Oleh karena itu, sangat sulit untuk mengatakan apakah seorang model "berhasil dengan baik." Misalnya, algoritme pengelompokan hipotetis

kami dapat mengelompokkan semua gambar yang menunjukkan wajah di profil dan semua gambar wajah penuh. Ini tentu akan menjadi cara yang mungkin untuk membagi koleksi gambar wajah orang, tetapi bukan itu yang kami cari. Namun, tidak ada cara bagi kita untuk "memberi tahu" algoritma apa yang kita cari, dan seringkali satu-satunya cara untuk mengevaluasi hasil dari algoritma yang tidak diawasi adalah dengan memeriksanya secara manual.

Akibatnya, algoritma tanpa pengawasan sering digunakan dalam pengaturan eksplorasi, ketika seorang ilmuwan data ingin memahami data dengan lebih baik, daripada sebagai bagian dari sistem otomatis yang lebih besar. Aplikasi umum lainnya untuk algoritma yang tidak diawasi adalah sebagai langkah pra-pemrosesan untuk algoritma yang diawasi. Mempelajari representasi data baru terkadang dapat meningkatkan akurasi algoritme yang diawasi, atau dapat menyebabkan pengurangan memori dan konsumsi waktu.

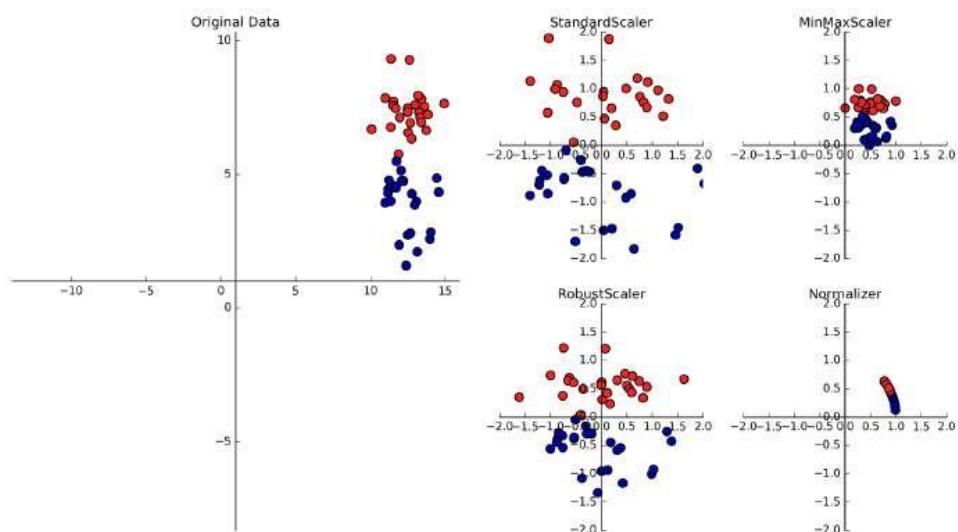
Sebelum kita mulai dengan algoritma "nyata" tanpa pengawasan, kita akan membahas secara singkat beberapa metode pra-pemrosesan sederhana yang sering berguna. Meskipun prapemrosesan dan penskalaan sering digunakan bersamaan dengan algoritme pembelajaran terawasi, metode penskalaan tidak menggunakan informasi terawasi, membuatnya tidak terawasi.

### 3.2 PRAPEMROSESAN DAN PENSKALAAN

Pada bab sebelumnya kita melihat bahwa beberapa algoritme, seperti jaringan saraf dan SVM, sangat sensitif terhadap penskalaan data. Oleh karena itu, praktik yang umum adalah menyesuaikan fitur sehingga representasi data lebih cocok untuk algoritma ini. Seringkali, ini adalah penskalaan ulang dan pergeseran data per fitur yang sederhana. Kode berikut (Gambar 3.1) menunjukkan contoh sederhana:

In[2]:

`mglearn.plots.plot_scaling()`



**Gambar 3.1** Berbagai cara untuk mengubah skala dan melakukan pra-proses kumpulan data

### Berbagai Jenis Pemrosesan Sebelumnya

Plot pertama pada Gambar 3.1 menunjukkan dataset klasifikasi dua kelas sintetis dengan dua fitur. Fitur pertama (nilai sumbu x) adalah antara 10 dan 15. Fitur kedua (nilai sumbu y) adalah antara sekitar 1 dan 9.

Empat plot berikut menunjukkan empat cara berbeda untuk mengubah data yang menghasilkan lebih banyak rentang standar. StandardScaler dalam scikit-learn memastikan bahwa untuk setiap fitur rata-ratanya adalah 0 dan variansnya adalah 1, membawa semua fitur ke besaran yang sama. Namun, penskalaan ini tidak memastikan nilai minimum dan maksimum tertentu untuk fitur tersebut. RobustScaler bekerja mirip dengan StandardScaler dalam memastikan properti statistik untuk setiap fitur yang menjamin bahwa mereka berada pada skala yang sama. Namun, RobustScaler menggunakan median dan kuartil,1 daripada mean dan varians. Hal ini membuat RobustScaler mengabaikan titik data yang sangat berbeda dari yang lain (seperti kesalahan pengukuran). Titik data ganjil ini juga disebut outlier, dan dapat menyebabkan masalah untuk teknik penskalaan lainnya.

MinMaxScaler, di sisi lain, menggeser data sedemikian rupa sehingga semua fitur persis antara 0 dan 1. Untuk dataset dua dimensi ini berarti semua data terkandung dalam persegi panjang yang dibuat oleh sumbu x antara 0 dan 1 dan sumbu y antara 0 dan 1.

Terakhir, Normalizer melakukan jenis rescaling yang sangat berbeda. Ini menskalakan setiap titik data sedemikian rupa sehingga vektor fitur memiliki panjang Euclidean 1. Dengan kata lain, ini memproyeksikan titik data pada lingkaran (atau bola, dalam kasus dimensi yang lebih tinggi) dengan radius 1. Ini berarti setiap data titik diskalakan dengan angka yang berbeda (dengan kebalikan dari panjangnya). Normalisasi ini sering digunakan ketika hanya arah (atau sudut) data yang penting, bukan panjang vektor fitur.

### 3.3 MENERAPKAN TRANSFORMASI DATA

Sekarang setelah kita melihat apa yang dilakukan oleh berbagai jenis transformasi, mari kita terapkan menggunakan scikit-learn. Kita akan menggunakan dataset kanker yang kita lihat di Bab 2. Metode pra-pemrosesan seperti scaler biasanya diterapkan sebelum menerapkan algoritme pembelajaran mesin yang diawasi. Sebagai contoh, katakanlah kita ingin menerapkan kernel SVM (SVC) ke dataset kanker, dan menggunakan MinMaxScaler untuk pra-pemrosesan data. Kita mulai dengan memuat dataset kita dan membaginya menjadi training set dan test set (kita membutuhkan training dan test set terpisah untuk mengevaluasi model terawasi yang akan kita buat setelah preprocessing):

**In[3]:**

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    random_state=1)
print(X_train.shape)
print(X_test.shape)
```

**Out[3]:**

```
(426, 30)
(143, 30)
```

Sebagai pengingat, dataset berisi 569 titik data, masing-masing diwakili oleh 30 pengukuran. Kami membagi dataset menjadi 426 sampel untuk set pelatihan dan 143 sampel untuk set pengujian. Seperti model terawasi yang kami buat sebelumnya, pertama-tama kami mengimpor kelas yang mengimplementasikan prapemrosesan, dan kemudian membuat instance-nya:

**In[4]:**

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

Kami kemudian menyesuaikan scaler menggunakan metode fit, yang diterapkan pada data pelatihan. Untuk Min MaxScaler, metode fit menghitung nilai minimum dan maksimum setiap fitur pada set pelatihan. Berbeda dengan pengklasifikasi dan regresor Bab 2, scaler hanya dilengkapi dengan data (X\_train) saat fit dipanggil, dan y\_train tidak digunakan:

**In[4]:**

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

Untuk menerapkan transformasi yang baru saja kita pelajari—yaitu, untuk benar-benar menskalakan data pelatihan—kita menggunakan metode transformasi scaler. Metode transformasi digunakan dalam scikit-learn setiap kali model mengembalikan representasi baru dari data:

**In[6]:**

```
# transform data
X_train_scaled = scaler.transform(X_train)
# print dataset properties before and after scaling
print("transformed shape: {}".format(X_train_scaled.shape))
print("per-feature minimum before scaling:\n {}".format(X_train.min(axis=0)))
print("per-feature maximum before scaling:\n {}".format(X_train.max(axis=0)))
print("per-feature minimum after scaling:\n {}".format(
    X_train_scaled.min(axis=0)))
print("per-feature maximum after scaling:\n {}".format(
    X_train_scaled.max(axis=0)))
```

**Out[6]:**

```

transformed shape: (426, 30)
per-feature minimum before scaling:
[ 6.98   9.71  43.79 143.50   0.05   0.02   0.     0.     0.11
  0.05   0.12   0.36   0.76   6.80   0.     0.     0.     0.
  0.01   0.     7.93  12.02  50.41  185.20   0.07   0.03   0.
  0.     0.16   0.06]
per-feature maximum before scaling:
[ 28.11   39.28  188.5  2501.0   0.16   0.29   0.43   0.2
  0.300   0.100   2.87   4.88   21.98  542.20   0.03   0.14
  0.400   0.050   0.06   0.03   36.04   49.54  251.20  4254.00
  0.220   0.940   1.17   0.29   0.58   0.15]
per-feature minimum after scaling:
[ 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.     0.     0.     0.     0.     0.     0.     0.]
per-feature maximum after scaling:
[ 1.     1.     1.     1.     1.     1.     1.     1.     1.
  1.     1.     1.     1.     1.     1.     1.     1.     1.]

```

Data yang diubah memiliki bentuk yang sama dengan data asli—fiturnya hanya digeser dan diskalakan. Anda dapat melihat bahwa semua fitur sekarang antara 0 dan 1, seperti yang diinginkan.

Untuk menerapkan SVM ke data yang diskalakan, kita juga perlu mengubah set pengujian. Ini sekali lagi dilakukan dengan memanggil metode transformasi, kali ini di X\_test:

**In[7]:**

```

# transform test data
X_test_scaled = scaler.transform(X_test)
# print test data properties after scaling
print("per-feature minimum after scaling:\n{}".format(X_test_scaled.min(axis=0)))
print("per-feature maximum after scaling:\n{}".format(X_test_scaled.max(axis=0)))

```

**Out[7]:**

```

per-feature minimum after scaling:
[ 0.034  0.023  0.031  0.011  0.141  0.044  0.     0.     0.154 -0.006
 -0.001  0.006  0.004  0.001  0.039  0.011  0.     0.     -0.032  0.007
  0.027  0.058  0.02   0.009  0.109  0.026  0.     0.     -0.     -0.002]
per-feature maximum after scaling:
[ 0.958  0.815  0.956  0.894  0.811  1.22   0.88   0.933  0.932  1.037
  0.427  0.498  0.441  0.284  0.487  0.739  0.767  0.629  1.337  0.391
  0.896  0.793  0.849  0.745  0.915  1.132  1.07   0.924  1.205  1.631]

```

Mungkin agak mengejutkan, Anda dapat melihat bahwa untuk set pengujian, setelah penskalaan, minimum dan maksimum bukan 0 dan 1. Beberapa fitur bahkan berada di luar kisaran 0-1! Penjelasannya adalah bahwa MinMaxScaler (dan semua scaler lainnya) selalu menerapkan transformasi yang sama persis ke pelatihan dan set pengujian. Ini berarti metode transformasi selalu mengurangkan minimum set pelatihan dan membaginya dengan rentang set pelatihan, yang mungkin berbeda dari minimum dan rentang untuk set pengujian.

### 3.4 PELATIHAN PENSKALAAN DAN DATA UJI DENGAN CARA YANG SAMA

Penting untuk menerapkan transformasi yang sama persis ke set pelatihan dan set pengujian agar model yang diawasi dapat bekerja pada set pengujian. Contoh berikut (Gambar 3.2) mengilustrasikan apa yang akan terjadi jika kita menggunakan minimum dan range dari test set sebagai gantinya:

In[8]:

```

from sklearn.datasets import make_blobs
# make synthetic data
X, _ = make_blobs(n_samples=50, centers=5, random_state=4, cluster_std=2)
# split it into training and test sets
X_train, X_test = train_test_split(X, random_state=5, test_size=.1)

# plot the training and test sets
fig, axes = plt.subplots(1, 3, figsize=(13, 4))

axes[0].scatter(X_train[:, 0], X_train[:, 1],
                 c=mlearn.cm2(0), label="Training set", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^',
                 c=mlearn.cm2(1), label="Test set", s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Original Data")

# scale the data using MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

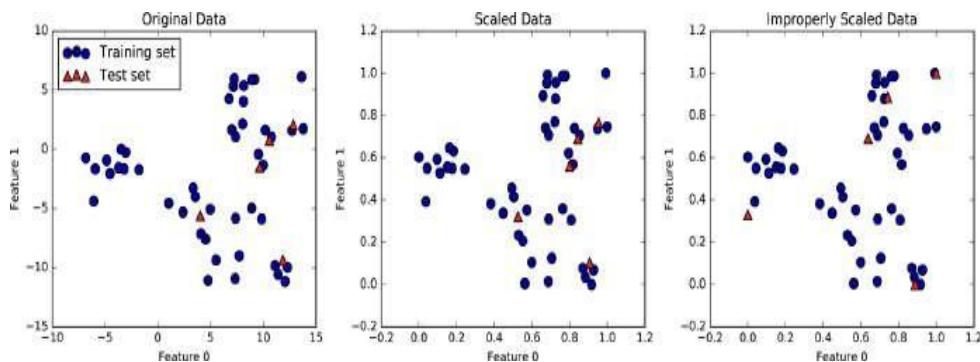
# visualize the properly scaled data
axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
                 c=mlearn.cm2(0), label="Training set", s=60)
axes[1].scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], marker='^',
                 c=mlearn.cm2(1), label="Test set", s=60)
axes[1].set_title("Scaled Data")

# rescale the test set separately
# so test set min is 0 and test set max is 1
# DO NOT DO THIS! For illustration purposes only.
test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)

# visualize wrongly scaled data
axes[2].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
                 c=mlearn.cm2(0), label="training set", s=60)
axes[2].scatter(X_test_scaled_badly[:, 0], X_test_scaled_badly[:, 1],
                 marker='^', c=mlearn.cm2(1), label="test set", s=60)
axes[2].set_title("Improperly Scaled Data")

for ax in axes:
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")

```



**Gambar 3.2** Pengaruh pelatihan penskalaan dan data uji ditampilkan di sebelah kiri bersama-sama (tengah) dan secara terpisah (kanan)

Panel pertama adalah set data dua dimensi yang tidak diskalakan, dengan set pelatihan ditampilkan sebagai lingkaran dan set pengujian ditampilkan sebagai segitiga. Panel kedua adalah data yang sama, tetapi diskalakan menggunakan MinMaxScaler. Di sini, kami memanggil fit pada set pelatihan, dan kemudian memanggil transformasi pada set pelatihan dan pengujian. Anda dapat melihat bahwa dataset di panel kedua terlihat identik dengan yang pertama; hanya kutu pada sumbu yang berubah. Sekarang semua fitur berada di antara 0 dan 1. Anda juga dapat melihat bahwa nilai fitur minimum dan maksimum untuk data uji (segitiga) bukan 0 dan 1.

Panel ketiga menunjukkan apa yang akan terjadi jika kita menskalakan set pelatihan dan set pengujian secara terpisah. Dalam hal ini, nilai fitur minimum dan maksimum untuk training dan test set adalah 0 dan 1. Namun sekarang dataset terlihat berbeda. Titik uji berpindah secara tidak selaras ke set pelatihan, karena skalanya berbeda. Kami mengubah susunan data dengan cara yang sewenang-wenang. Jelas ini bukan yang ingin kami lakukan.

Sebagai cara lain untuk memikirkan hal ini, bayangkan set pengujian Anda adalah satu titik. Tidak ada cara untuk menskalakan satu titik dengan benar, untuk memenuhi persyaratan minimum dan maksimum dari MinMaxScaler. Tetapi ukuran set pengujian Anda seharusnya tidak mengubah pemrosesan Anda.

### Jalan Pintas dan Alternatif yang Efisien

Seringkali, Anda ingin menyesuaikan model pada beberapa kumpulan data, lalu mengubahnya. Ini adalah tugas yang sangat umum, yang seringkali dapat dihitung lebih efisien daripada hanya dengan memanggil fit dan kemudian mentransformasikannya. Untuk use case ini, semua model yang memiliki metode transform juga memiliki metode fit\_transform. Berikut adalah contoh menggunakan StandardScaler:

**In[9]:**

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# calling fit and transform in sequence (using method chaining)
X_scaled = scaler.fit(X).transform(X)
# same result, but more efficient computation
X_scaled_d = scaler.fit_transform(X)
```

Meskipun fit\_transform tidak selalu lebih efisien untuk semua model, masih merupakan praktik yang baik untuk menggunakan metode ini ketika mencoba mengubah set pelatihan.

## 3.5 PENGARUH PREPROCESSING PADA PEMBELAJARAN TERAWASI

Sekarang mari kembali ke kumpulan data kanker dan lihat efek penggunaan MinMaxScaler dalam mempelajari SVC (ini adalah cara berbeda untuk melakukan penskalaan yang sama seperti yang kita lakukan di Bab 2). Pertama, mari kita pasang SVC pada data asli lagi untuk perbandingan:

**In[10]:**

```
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    random_state=0)

svm = SVC(C=100)
svm.fit(X_train, y_train)
print("Test set accuracy: {:.2f}".format(svm.score(X_test, y_test)))
```

**Out[10]:**

```
Test set accuracy: 0.63
```

Sekarang, mari skalakan data menggunakan MinMaxScaler sebelum memasang SVC:

**In[11]:**

```
# preprocessing using 0-1 scaling
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)

# scoring on the scaled test set
print("Scaled test set accuracy: {:.2f}".format(
    svm.score(X_test_scaled, y_test)))
```

**Out[11]:**

```
Scaled test set accuracy: 0.97
```

Seperti yang kita lihat sebelumnya, efek penskalaan data cukup signifikan. Meskipun penskalaan data tidak melibatkan matematika yang rumit, praktik yang baik adalah menggunakan mekanisme penskalaan yang disediakan oleh scikit-learn daripada menerapkannya sendiri, karena mudah membuat kesalahan bahkan dalam perhitungan sederhana ini.

Anda juga dapat dengan mudah mengganti satu algoritme prapemrosesan dengan yang lain dengan mengubah kelas yang Anda gunakan, karena semua kelas prapemrosesan memiliki antarmuka yang sama, yang terdiri dari metode fit and transform:

**In[12]:**

```
# preprocessing using zero mean and unit variance scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)

# scoring on the scaled test set
print("SVM test accuracy: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

**Out[12]:**

```
SVM test accuracy: 0.96
```

Sekarang kita telah melihat bagaimana transformasi data sederhana untuk pra-pemrosesan bekerja, mari beralih ke transformasi yang lebih menarik menggunakan pembelajaran tanpa pengawasan.

### 3.6 PENGURANGAN DIMENSI, EKSTRAKSI FITUR, DAN PEMBELAJARAN MANIFOLD

Seperti yang telah kita bahas sebelumnya, mengubah data menggunakan pembelajaran tanpa pengawasan dapat memiliki banyak motivasi. Motivasi yang paling umum adalah visualisasi, kompresi data, dan menemukan representasi yang lebih informatif untuk diproses lebih lanjut.

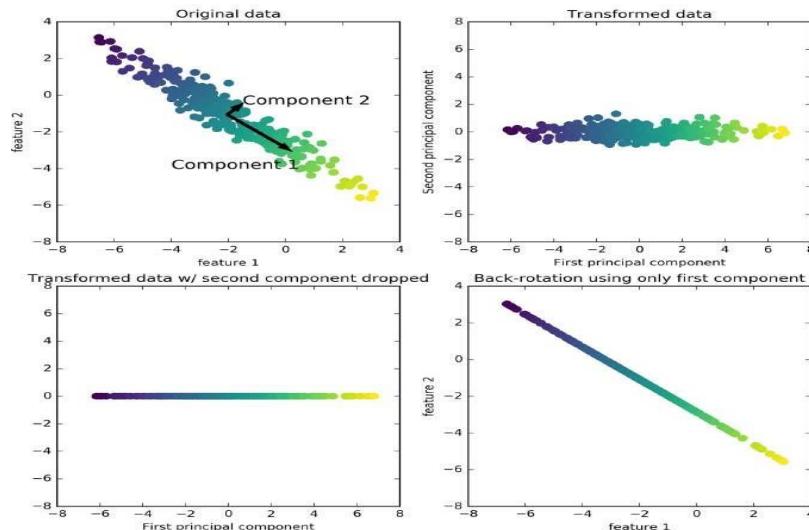
Salah satu algoritma yang paling sederhana dan paling banyak digunakan untuk semua ini adalah analisis komponen utama. Kami juga akan melihat dua algoritma lain: faktorisasi matriks non-negatif (NMF), yang biasa digunakan untuk ekstraksi fitur, dan t-SNE, yang biasa digunakan untuk visualisasi menggunakan plot sebar dua dimensi.

### 3.7 ANALISIS KOMPONEN UTAMA (PCA)

Analisis komponen utama adalah metode yang memutar kumpulan data sedemikian rupa sehingga fitur yang diputar tidak berkorelasi secara statistik. Rotasi ini sering diikuti dengan memilih hanya sebagian dari fitur baru, menurut seberapa penting fitur tersebut untuk menjelaskan data. Contoh berikut (Gambar 3.3) mengilustrasikan efek PCA pada dataset dua dimensi sintetik:

**In[13]:**

```
mglearn.plots.plot_pca_illustration()
```



**Gambar 3.3** Transformasi data dengan PCA

Plot pertama (kiri atas) menunjukkan titik data asli, diwarnai untuk membedakannya. Algoritme melanjutkan dengan terlebih dahulu menemukan arah varians maksimum, berlabel "Komponen 1." Ini adalah arah (atau vektor) dalam data yang berisi sebagian besar informasi, atau dengan kata lain, arah di mana fitur-fiturnya paling berkorelasi satu sama lain. Kemudian, algoritme menemukan arah yang berisi informasi paling banyak sambil ortogonal (pada sudut siku-siku) ke arah pertama. Dalam dua dimensi, hanya ada satu kemungkinan orientasi yang berada pada sudut siku-siku, tetapi dalam ruang dimensi yang lebih tinggi akan ada (tak terhingga) banyak arah ortogonal. Meskipun kedua komponen digambar sebagai panah, tidak masalah di mana kepala dan ekor berada; kita bisa menggambar komponen pertama dari tengah ke kiri atas, bukan ke kanan bawah. Arah yang ditemukan menggunakan proses ini disebut komponen utama, karena merupakan arah utama varians dalam data. Secara umum, ada banyak komponen utama sebagai fitur asli.

Plot kedua (kanan atas) menunjukkan data yang sama, tetapi sekarang diputar sehingga komponen utama pertama sejajar dengan sumbu x dan komponen utama kedua sejajar dengan sumbu y. Sebelum rotasi, mean dikurangkan dari data, sehingga data yang diubah dipusatkan di sekitar nol. Pada representasi terputar yang ditemukan oleh PCA, kedua sumbu tersebut tidak berkorelasi, artinya matriks korelasi data pada representasi ini adalah nol kecuali diagonalnya.

Kita dapat menggunakan PCA untuk pengurangan dimensi dengan mempertahankan hanya beberapa komponen utama. Dalam contoh ini, kita mungkin hanya menyimpan komponen utama pertama, seperti yang ditunjukkan pada panel ketiga pada Gambar 3.3 (kiri bawah). Ini mengurangi data dari kumpulan data dua dimensi menjadi kumpulan data satu dimensi. Namun, perhatikan bahwa alih-alih hanya menyimpan salah satu fitur asli, kami menemukan arah yang paling menarik (kiri atas ke kanan bawah di panel pertama) dan mempertahankan arah ini, komponen utama pertama.

Akhirnya, kita dapat membatalkan rotasi dan menambahkan mean kembali ke data. Ini akan menghasilkan data yang ditunjukkan pada panel terakhir pada Gambar 3.3. Titik-titik ini berada di ruang fitur asli, tetapi kami hanya menyimpan informasi yang terkandung dalam komponen utama pertama. Transformasi ini terkadang digunakan untuk menghilangkan efek noise dari data atau memvisualisasikan bagian mana dari informasi yang dipertahankan menggunakan komponen utama.

### **Menerapkan PCA ke dataset kanker untuk visualisasi**

Salah satu aplikasi PCA yang paling umum adalah memvisualisasikan kumpulan data berdimensi tinggi. Seperti yang kita lihat di Bab 1, sulit untuk membuat plot pencar data yang memiliki lebih dari dua fitur. Untuk dataset Iris, kami dapat membuat plot pasangan (Gambar 1-3 di Bab 1) yang memberi kami gambaran sebagian data dengan menunjukkan kepada kami semua kemungkinan kombinasi dari dua fitur. Tetapi jika kita ingin melihat dataset Kanker Payudara, bahkan menggunakan plot berpasangan itu rumit. Kumpulan data ini memiliki 30 fitur, yang akan menghasilkan  $30 * 14 = 420$  plot pencar! Kami tidak akan pernah bisa melihat semua plot ini secara detail, apalagi mencoba memahaminya.

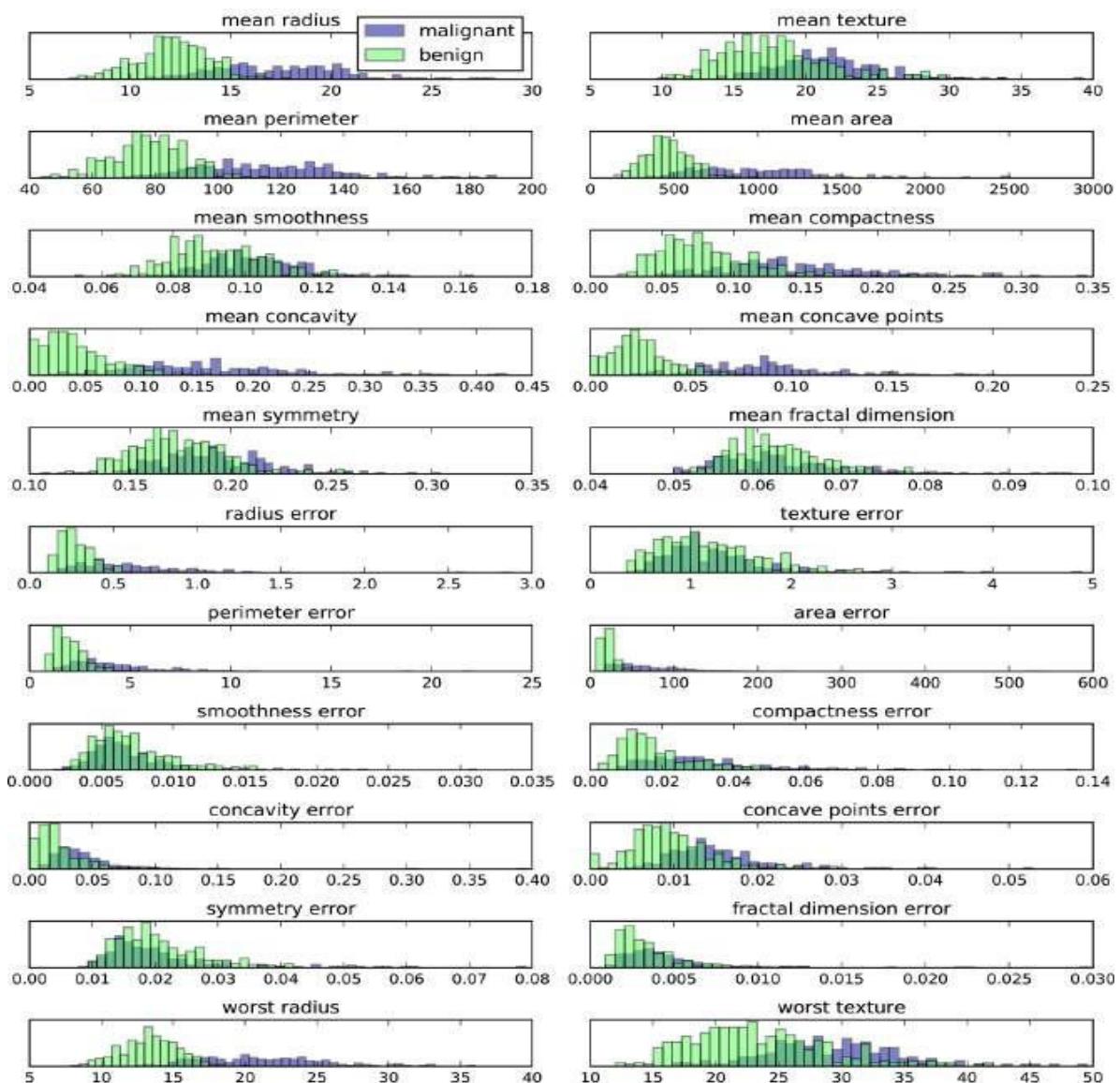
Ada visualisasi yang lebih sederhana yang dapat kita gunakan—menghitung histogram dari masing-masing fitur untuk dua kelas, kanker jinak dan ganas (Gambar 3.4):

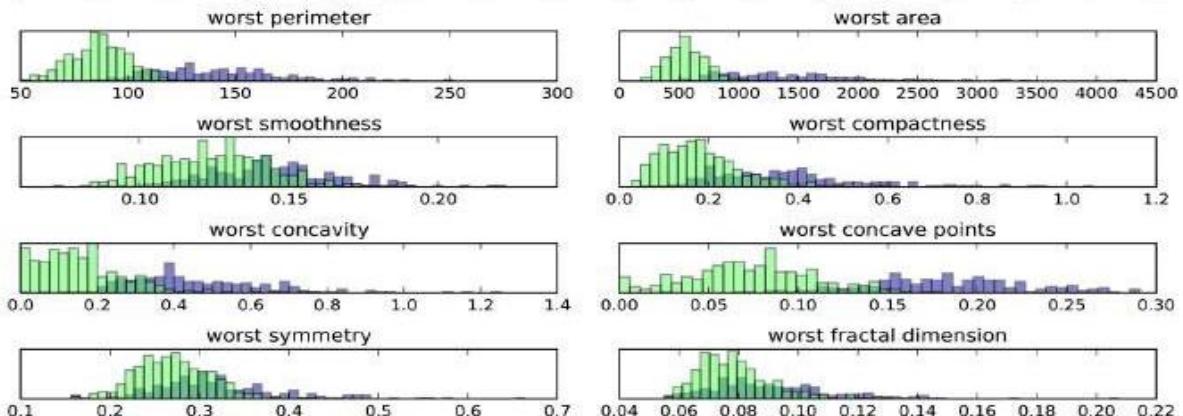
In[14]:

```
fig, axes = plt.subplots(15, 2, figsize=(10, 20))
malignant = cancer.data[cancer.target == 0]
benign = cancer.data[cancer.target == 1]

ax = axes.ravel()

for i in range(30):
    _, bins = np.histogram(cancer.data[:, i], bins=50)
    ax[i].hist(malignant[:, i], bins=bins, color=mglearn.cm3(0), alpha=.5)
    ax[i].hist(benign[:, i], bins=bins, color=mglearn.cm3(2), alpha=.5)
    ax[i].set_title(cancer.feature_names[i])
    ax[i].set_yticks(())
    ax[0].set_xlabel("Feature magnitude")
    ax[0].set_ylabel("Frequency")
    ax[0].legend(["malignant", "benign"], loc="best")
fig.tight_layout()
```





**Gambar 3.4 Histogram fitur per kelas pada dataset Kanker Payudara**

Di sini kita membuat histogram untuk setiap fitur, menghitung seberapa sering titik data muncul dengan fitur dalam rentang tertentu (disebut bin). Setiap plot melapisi dua histogram, satu untuk semua titik di kelas jinak (biru) dan satu untuk semua titik di kelas ganas (merah). Ini memberi kita beberapa gambaran tentang bagaimana setiap fitur didistribusikan di dua kelas, dan memungkinkan kita untuk menebak fitur mana yang lebih baik dalam membedakan sampel ganas dan jinak. Misalnya, fitur ‐kehalusan kesalahan‐ tampaknya kurang informatif, karena dua histogram sebagian besar tumpang tindih, sedangkan fitur ‐titik cekung terburuk‐ tampaknya cukup informatif, karena histogramnya cukup terputus-putus.

Namun, plot ini tidak menunjukkan kepada kita apa pun tentang interaksi antara variabel dan bagaimana ini berhubungan dengan kelas. Menggunakan PCA, kita dapat menangkap interaksi utama dan mendapatkan gambaran yang sedikit lebih lengkap. Kita dapat menemukan dua komponen utama pertama, dan memvisualisasikan data dalam ruang dua dimensi baru ini dengan plot pencar tunggal.

Sebelum kami menerapkan PCA, kami menskalakan data kami sehingga setiap fitur memiliki varian unit menggunakan Penskala Standar:

**In[15]:**

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

Mempelajari transformasi PCA dan menerapkannya semudah menerapkan transformasi pra-pemrosesan. Kami membuat instance objek PCA, menemukan komponen utama dengan memanggil metode fit, dan kemudian menerapkan pengurangan rotasi dan dimensi dengan memanggil transformasi. Secara default, PCA hanya memutar (dan menggeser) data, tetapi menyimpan semua komponen utama. Untuk mengurangi dimensi data, kita perlu menentukan berapa banyak komponen yang ingin kita simpan saat membuat objek PCA:

In[16]:

```
from sklearn.decomposition import PCA
# keep the first two principal components of the data
pca = PCA(n_components=2)
# fit PCA model to breast cancer data
pca.fit(X_scaled)

# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: {}".format(str(X_scaled.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
```

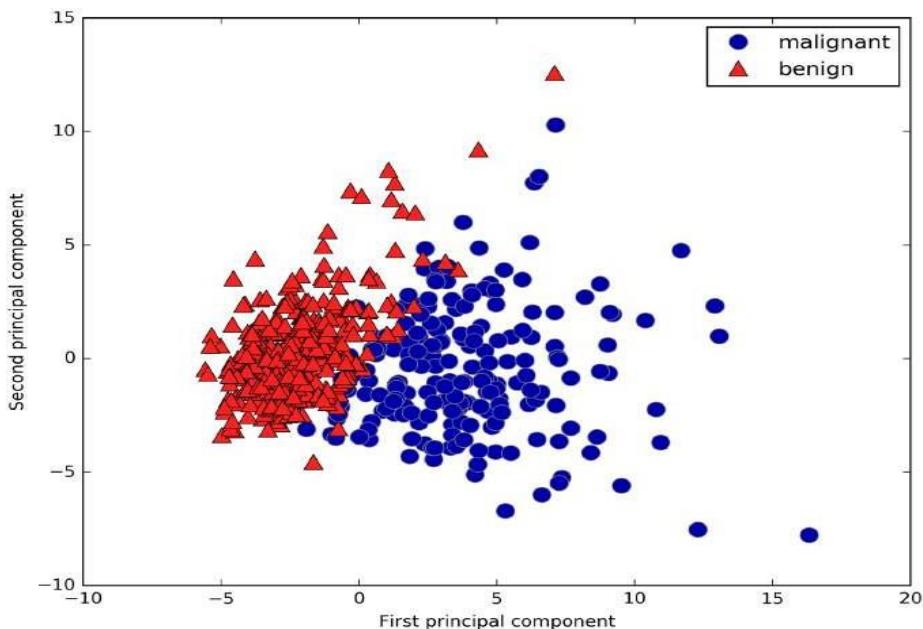
Out[16]:

```
Original shape: (569, 30)
Reduced shape: (569, 2)
```

Sekarang kita dapat memplot dua komponen utama pertama (Gambar 3.5):

In[17]:

```
# plot first vs. second principal component, colored by class
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
plt.legend(cancer.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```



**Gambar 3.5** Plot sebar dua dimensi dari dataset Kanker Payudara menggunakan dua komponen utama pertama

Penting untuk dicatat bahwa PCA adalah metode tanpa pengawasan, dan tidak menggunakan informasi kelas apa pun saat menemukan rotasi. Ini hanya melihat korelasi dalam data. Untuk plot sebar yang ditunjukkan di sini, kami memplot komponen utama

pertama terhadap komponen utama kedua, dan kemudian menggunakan informasi kelas untuk mewarnai titik-titiknya. Anda dapat melihat bahwa dua kelas terpisah cukup baik dalam ruang dua dimensi ini. Ini membuat kita percaya bahwa bahkan pengklasifikasi linier (yang akan mempelajari garis dalam ruang ini) dapat melakukan pekerjaan yang cukup baik dalam membedakan kedua kelas. Kita juga dapat melihat bahwa titik-titik ganas (merah) lebih menyebar daripada titik-titik jinak (biru) —sesuatu yang sudah dapat kita lihat sedikit dari histogram pada Gambar 3.4.

Kelemahan PCA adalah bahwa dua sumbu dalam plot seringkali tidak mudah untuk ditafsirkan. Komponen utama sesuai dengan arah dalam data asli, jadi mereka adalah kombinasi dari fitur asli. Namun, kombinasi ini biasanya sangat kompleks, seperti yang akan segera kita lihat. Komponen utama itu sendiri disimpan dalam atribut `component_` dari objek PCA selama pemasangan:

**In[18]:**

```
print("PCA component shape: {}".format(pca.components_.shape))
```

**Out[18]:**

```
PCA component shape: (2, 30)
```

Setiap baris dalam `komponen_` sesuai dengan satu komponen utama, dan mereka diurutkan berdasarkan kepentingannya (komponen utama pertama didahului, dll.). Kolom sesuai dengan atribut fitur asli PCA dalam contoh ini, "radius rata-rata", "tekstur rata-rata", dan seterusnya. Mari kita lihat isi dari `component_`:

**In[19]:**

```
print("PCA components:\n{}".format(pca.components_))
```

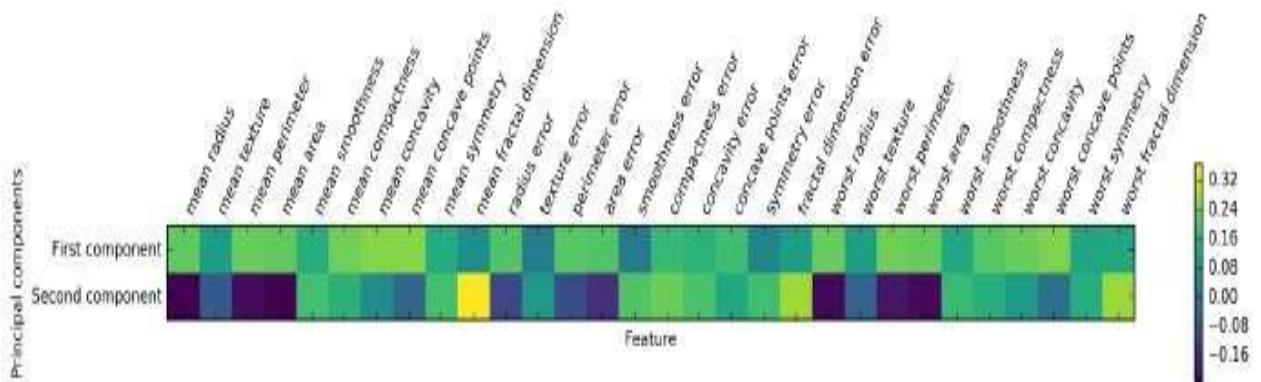
**Out[19]:**

```
PCA components:
[[ 0.219  0.104  0.228  0.221  0.143  0.239  0.258  0.261  0.138  0.064
   0.206  0.017  0.211  0.203  0.015  0.17   0.154  0.183  0.042  0.103
   0.228  0.104  0.237  0.225  0.128  0.21   0.229  0.251  0.123  0.132]
 [-0.234 -0.06  -0.215 -0.231  0.186  0.152  0.06  -0.035  0.19   0.367
  -0.106  0.09  -0.089 -0.152  0.204  0.233  0.197  0.13   0.184  0.28
  -0.22   -0.045 -0.2   -0.219  0.172  0.144  0.098 -0.008  0.142  0.275]]
```

Kita juga dapat memvisualisasikan koefisien menggunakan peta panas (Gambar 3.6), yang mungkin lebih mudah dipahami:

**In[20]:**

```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["First component", "Second component"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),
           cancer.feature_names, rotation=60, ha='left')
plt.xlabel("Feature")
plt.ylabel("Principal components")
```



**Gambar 3.6** Peta panas dari dua komponen utama pertama pada dataset Kanker Payudara

Anda dapat melihat bahwa di komponen pertama, semua fitur memiliki tanda yang sama (negatif, tetapi seperti yang kami sebutkan sebelumnya, tidak masalah ke arah mana panah menunjuk). Itu berarti ada korelasi umum antara semua fitur. Karena satu pengukuran tinggi, yang lain cenderung tinggi juga. Komponen kedua memiliki tanda-tanda campuran, dan kedua komponen melibatkan semua 30 fitur. Pencampuran semua fitur inilah yang membuat penjelasan sumbu pada Gambar 3.6 begitu rumit.

#### Eigenfaces untuk ekstraksi fitur

Aplikasi lain dari PCA yang kami sebutkan sebelumnya adalah ekstraksi fitur. Gagasan di balik ekstraksi fitur adalah memungkinkan untuk menemukan representasi data Anda yang lebih cocok untuk dianalisis daripada representasi mentah yang diberikan kepada Anda. Contoh yang bagus dari aplikasi di mana ekstraksi fitur sangat membantu adalah dengan gambar. Gambar terdiri dari piksel, biasanya disimpan sebagai intensitas merah, hijau, dan biru (RGB). Objek dalam gambar biasanya terdiri dari ribuan piksel, dan hanya bersama-sama mereka bermakna.

Kami akan memberikan aplikasi ekstraksi fitur yang sangat sederhana pada gambar menggunakan PCA, dengan bekerja dengan gambar wajah dari dataset Labeled Faces in the Wild. Kumpulan data ini berisi gambar wajah selebritas yang diunduh dari Internet, dan termasuk wajah politisi, penyanyi, aktor, dan atlet dari awal 2000-an. Kami menggunakan versi skala abu-abu dari gambar-gambar ini, dan menurunkannya untuk pemrosesan yang lebih cepat. Anda dapat melihat beberapa gambar pada Gambar 3.7:

**In[21]:**

```
from sklearn.datasets import fetch_lfw_people
people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
image_shape = people.images[0].shape

fix, axes = plt.subplots(2, 5, figsize=(15, 8),
                       subplot_kw={'xticks': (), 'yticks': ()})
for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image)
    ax.set_title(people.target_names[target])
```



**Gambar 3.7** Beberapa gambar dari kumpulan data Labeled Faces in the Wild

Ada 3.023 gambar, masing-masing berukuran 87x65 piksel, milik 62 orang yang berbeda:

**In[22]:**

```
print("people.images.shape: {}".format(people.images.shape))
print("Number of classes: {}".format(len(people.target_names)))
```

**Out[22]:**

```
people.images.shape: (3023, 87, 65)
Number of classes: 62
```

Namun, kumpulan datanya agak miring, berisi banyak gambar George W. Bush dan Colin Powell, seperti yang Anda lihat di sini:

**In[23]:**

```
# count how often each target appears
counts = np.bincount(people.target)
# print counts next to target names
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print("{}: {} ({})".format(name, count, count % 3))
    if (i + 1) % 3 == 0:
        print()
```

**Out[23]:**

Alejandro Toledo	39	Alvaro Uribe	35
Amelie Mauresmo	21	Andre Agassi	36
Angelina Jolie	20	Arnold Schwarzenegger	42
Atal Bihari Vajpayee	24	Bill Clinton	29
Carlos Menem	21	Colin Powell	236
David Beckham	31	Donald Rumsfeld	121
George W Bush	530	George Robertson	22
Gerhard Schroeder	109	Gloria Macapagal Arroyo	44
Gray Davis	26	Guillermo Coria	30
Hamid Karzai	22	Hans Blix	39
Hugo Chavez	71	Igor Ivanov	20
[...]		[...]	
Laura Bush	41	Lindsay Davenport	22
Lleyton Hewitt	41	Luiz Inacio Lula da Silva	48
Mahmoud Abbas	29	Megawati Sukarnoputri	33
Michael Bloomberg	20	Naomi Watts	22
Nestor Kirchner	37	Paul Bremer	20
Pete Sampras	22	Recep Tayyip Erdogan	30
Ricardo Lagos	27	Roh Moo-hyun	32
Rudolph Giuliani	26	Saddam Hussein	23
Serena Williams	52	Silvio Berlusconi	33
Tiger Woods	23	Tom Daschle	25
Tom Ridge	33	Tony Blair	144
Vicente Fox	32	Vladimir Putin	49
Winona Ryder	24		

Untuk membuat data kurang miring, kami hanya akan mengambil hingga 50 gambar dari setiap orang (jika tidak, ekstraksi fitur akan kewalahan oleh kemungkinan George W.Bush):

**In[24]:**

```
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1

X_people = people.data[mask]
y_people = people.target[mask]

# scale the grayscale values to be between 0 and 1
# instead of 0 and 255 for better numeric stability
X_people = X_people / 255.
```

Tugas umum dalam pengenalan wajah adalah menanyakan apakah wajah yang sebelumnya tidak terlihat milik orang yang dikenal dari database. Ini memiliki aplikasi dalam koleksi foto, media sosial, dan aplikasi keamanan. Salah satu cara untuk memecahkan masalah ini adalah dengan membangun classifier di mana setiap orang adalah kelas yang terpisah. Namun, biasanya ada banyak orang yang berbeda dalam database wajah, dan sangat sedikit gambar dari orang yang sama (yaitu, sangat sedikit contoh pelatihan per kelas). Itu membuatnya sulit untuk melatih sebagian besar pengklasifikasi. Selain itu, Anda sering ingin dapat menambahkan orang baru dengan mudah, tanpa perlu melatih ulang model besar.

Solusi sederhana adalah dengan menggunakan pengklasifikasi satu-tetangga-terdekat yang mencari gambar wajah yang paling mirip dengan wajah yang Anda klasifikasikan. Pengklasifikasi ini pada prinsipnya dapat bekerja hanya dengan satu contoh pelatihan per kelas. Mari kita lihat seberapa baik kinerja KNeighborsClassifier di sini:

**In[25]:**

```
from sklearn.neighbors import KNeighborsClassifier
# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test, y_test)))
```

**Out[25]:**

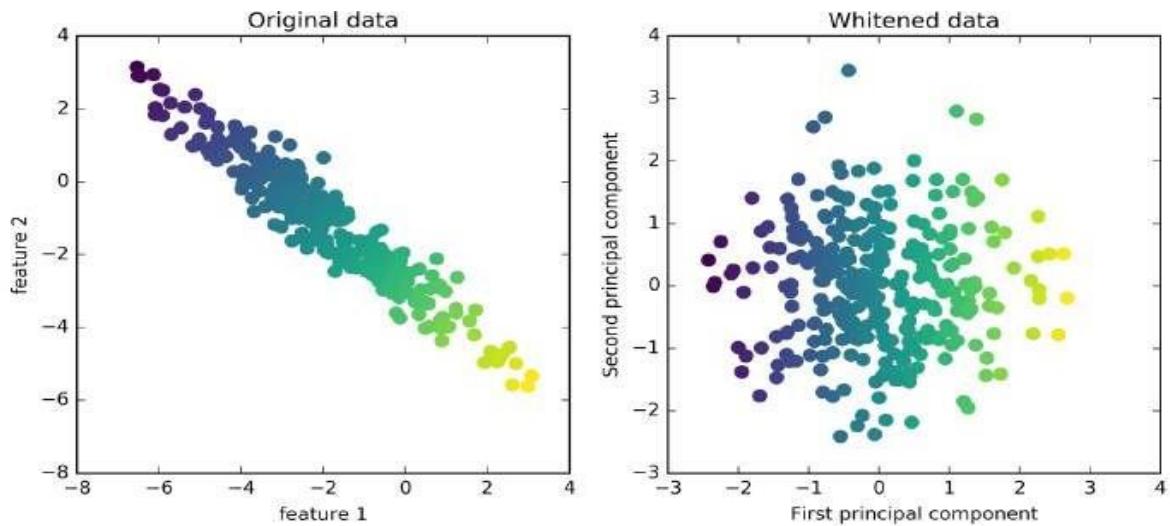
```
Test set score of 1-nn: 0.27
```

Kami memperoleh akurasi 26,6%, yang sebenarnya tidak terlalu buruk untuk masalah klasifikasi kelas 62 (penebakan acak akan memberi Anda sekitar  $1/62 =$  akurasi 1,5%), tetapi juga tidak bagus. Kami hanya mengidentifikasi seseorang dengan benar setiap keempat kalinya.

Di sinilah PCA masuk. Menghitung jarak dalam ruang piksel asli adalah cara yang cukup buruk untuk mengukur kesamaan antar wajah. Saat menggunakan representasi piksel untuk membandingkan dua gambar, kami membandingkan nilai skala abu-abu dari setiap piksel individu dengan nilai piksel pada posisi yang sesuai pada gambar lainnya. Representasi ini sangat berbeda dari bagaimana manusia akan menafsirkan citra wajah, dan sulit untuk menangkap fitur wajah menggunakan representasi mentah ini. Misalnya, menggunakan jarak di sepanjang komponen utama dapat meningkatkan akurasi kami. Di sini, kami mengaktifkan opsi pemutihan PCA, yang mengubah skala komponen utama agar memiliki skala yang sama. Ini sama dengan menggunakan StandardScaler setelah transformasi. Menggunakan kembali data dari Gambar 3.3 lagi, memutihkan tidak hanya berhubungan dengan memutar data, tetapi juga mengubah skalanya sehingga panel tengah adalah lingkaran, bukan elips (lihat Gambar 3.8):

**In[26]:**

```
mglearn.plots.plot_pca_whitening()
```



**Gambar 3.8** Transformasi data dengan PCA menggunakan pemutih

Kami menyesuaikan objek PCA dengan data pelatihan dan mengekstrak 100 komponen utama pertama. Kemudian kami mengubah data pelatihan dan pengujian:

**In[27]:**

```
pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print("X_train_pca.shape: {}".format(X_train_pca.shape))
```

**Out[27]:**

```
X_train_pca.shape: (1537, 100)
```

Data baru memiliki 100 fitur, 100 komponen utama pertama. Sekarang, kita dapat menggunakan representasi baru untuk mengklasifikasikan gambar kita menggunakan pengklasifikasi satu-tetangga-terdekat:

**In[28]:**

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("Test set accuracy: {:.2f}".format(knn.score(X_test_pca, y_test)))
```

**Out[28]:**

```
Test set accuracy: 0.36
```

Akurasi kami meningkat cukup signifikan, dari 26,6% menjadi 35,7%, membenarkan intuisi kami bahwa komponen utama mungkin memberikan representasi data yang lebih baik.

Untuk data citra, kita juga dapat dengan mudah memvisualisasikan komponen utama yang ditemukan. Ingat bahwa komponen sesuai dengan arah di ruang input. Ruang input di sini adalah gambar skala abu-abu 50x37 piksel, jadi arah dalam ruang ini juga merupakan gambar skala abu-abu 50x37 piksel. Mari kita lihat beberapa komponen utama pertama (Gambar 3.9):

In[29]:

```
print("pca.components_.shape: {}".format(pca.components_.shape))
```

Out[29]:

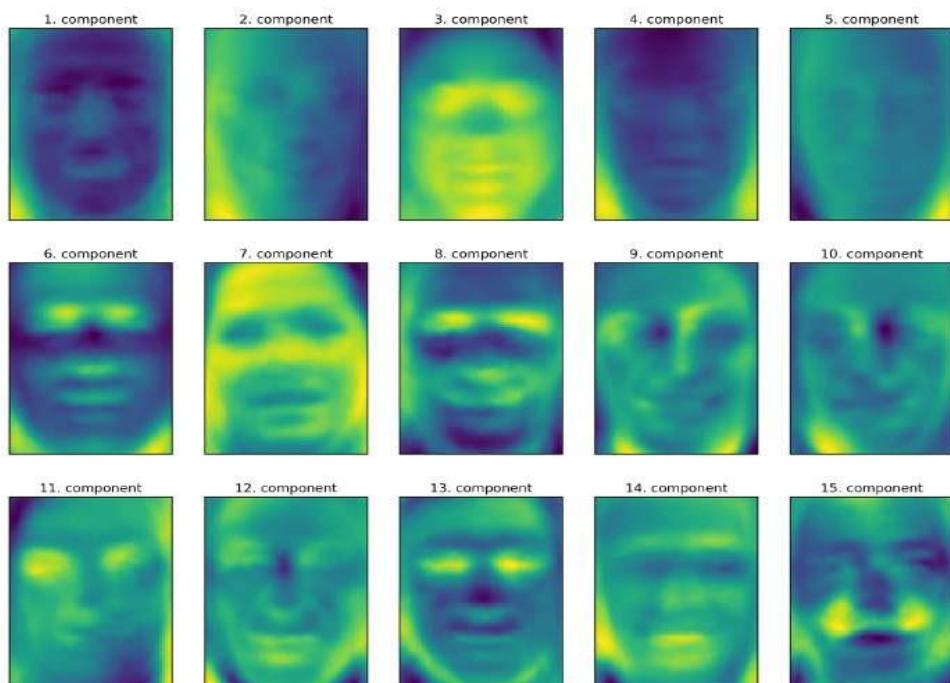
```
pca.components_.shape: (100, 5655)
```

In[30]:

```
fix, axes = plt.subplots(3, 5, figsize=(15, 12),
                       subplot_kw={'xticks': (), 'yticks': ()})
for i, (component, ax) in enumerate(zip(pca.components_, axes.ravel())):
    ax.imshow(component.reshape(image_shape),
              cmap='viridis')
    ax.set_title("{} component".format(i + 1))
```

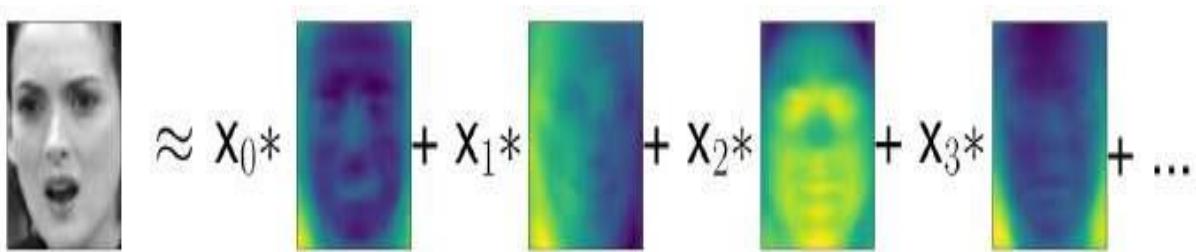
Meskipun kami tentu saja tidak dapat memahami semua aspek dari komponen ini, kami dapat menebak aspek mana dari gambar wajah yang ditangkap oleh beberapa komponen. Komponen pertama tampaknya sebagian besar mengkodekan kontras antara wajah dan latar belakang, komponen kedua mengkodekan perbedaan pencahayaan antara bagian kanan dan kiri wajah, dan seterusnya. Meskipun representasi ini sedikit lebih semantik daripada nilai piksel mentah, itu masih cukup jauh dari cara manusia memandang wajah.

Karena model PCA didasarkan pada piksel, perataan wajah (posisi mata, dagu, dan hidung) dan pencahayaan memiliki pengaruh kuat pada seberapa mirip dua gambar dalam representasi pikselnya. Tapi keselarasan dan pencahayaan mungkin bukan yang pertama kali dirasakan manusia. Saat meminta orang untuk menilai kesamaan wajah, mereka lebih cenderung menggunakan atribut seperti usia, jenis kelamin, ekspresi wajah, dan gaya rambut, yang merupakan atribut yang sulit disimpulkan dari intensitas piksel. Penting untuk diingat bahwa algoritme sering kali menginterpretasikan data (khususnya data visual, seperti gambar, yang sangat dikenal manusia) dengan cara yang sangat berbeda dari yang dilakukan manusia.



**Gambar 3.9** Vektor komponen dari 15 komponen utama pertama dari kumpulan data wajah  
Machine Learning (Dr. Budi Raharjo)

Mari kembali ke kasus spesifik PCA. Kami memperkenalkan transformasi PCA sebagai memutar data dan kemudian menjatuhkan komponen dengan varians rendah. Interpretasi lain yang berguna adalah mencoba menemukan beberapa angka (nilai fitur baru setelah rotasi PCA) sehingga kita dapat menyatakan titik uji sebagai jumlah bobot komponen utama (lihat Gambar 3.10).



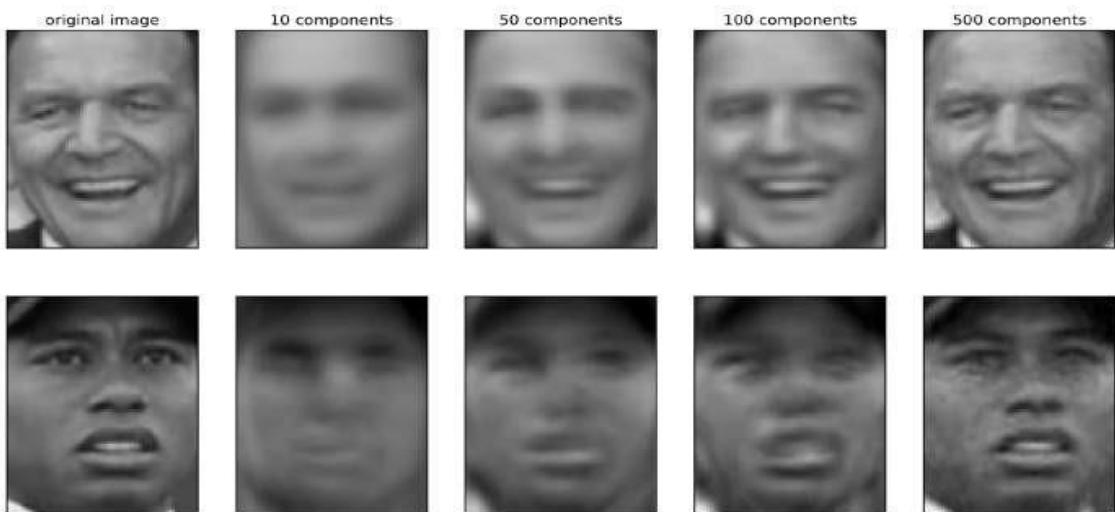
**Gambar 3.10** Tampilan skematis PCA sebagai penguraian gambar menjadi jumlah komponen yang tertimbang

Di sini,  $x_0$ ,  $x_1$ , dan seterusnya adalah koefisien komponen utama untuk titik data ini; dengan kata lain, mereka adalah representasi dari gambar dalam ruang yang diputar.

Cara lain yang dapat kita coba untuk memahami apa yang dilakukan model PCA adalah dengan melihat rekonstruksi data asli hanya menggunakan beberapa komponen. Pada Gambar 3.3, setelah menjatuhkan komponen kedua dan tiba di panel ketiga, kami membuka putaran dan menambahkan rata-rata kembali untuk mendapatkan titik baru di ruang asli dengan komponen kedua dihapus, seperti yang ditunjukkan pada panel terakhir. Kita dapat melakukan transformasi serupa untuk wajah dengan mereduksi data menjadi hanya beberapa komponen utama dan kemudian memutar kembali ke ruang aslinya. Pengembalian ke ruang fitur asli ini dapat dilakukan dengan menggunakan metode `inverse_transform`. Di sini, kami memvisualisasikan rekonstruksi beberapa wajah menggunakan 10, 50, 100, 500, atau 2.000 komponen (Gambar 3.11):

**In[32]:**

```
mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)
```





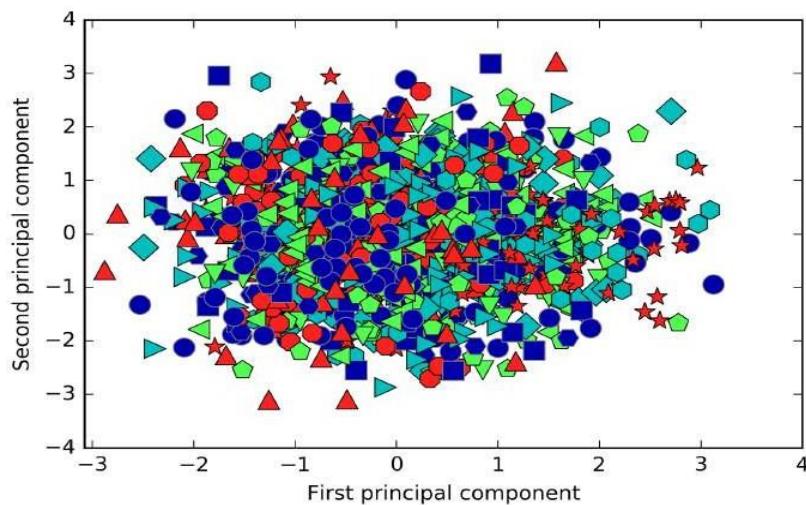
**Gambar 3.11** Merekonstruksi tiga gambar wajah menggunakan semakin banyak komponen utama

Anda dapat melihat bahwa ketika kami hanya menggunakan 10 komponen utama pertama, hanya esensi gambar, seperti orientasi wajah dan pencahayaan, yang ditangkap. Dengan menggunakan semakin banyak komponen utama, semakin banyak detail dalam gambar yang dipertahankan. Ini sesuai dengan memperluas jumlah pada Gambar 3.10 untuk memasukkan lebih banyak istilah. Menggunakan komponen sebanyak piksel berarti kami tidak akan membuang informasi apa pun setelah rotasi, dan kami akan merekonstruksi gambar dengan sempurna.

Kami juga dapat mencoba menggunakan PCA untuk memvisualisasikan semua wajah dalam kumpulan data dalam plot pencar menggunakan dua komponen utama pertama (Gambar 3.12), dengan kelas yang diberikan oleh siapa yang ditunjukkan pada gambar, mirip dengan apa yang kami lakukan untuk kumpulan data kanker:

**In[33]:**

```
mglearn.discrete_scatter(X_train_pca[:, 0], X_train_pca[:, 1], y_train)
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```



**Gambar 3.12** Plot sebar dari dataset wajah menggunakan dua komponen utama pertama  
(lihat **Gambar 3.5** untuk gambar yang sesuai untuk dataset kanker)

Seperti yang Anda lihat, ketika kami hanya menggunakan dua komponen utama pertama, seluruh data hanyalah gumpalan besar, tanpa pemisahan kelas yang terlihat. Ini tidak terlalu mengejutkan, mengingat bahkan dengan 10 komponen, seperti yang ditunjukkan sebelumnya pada Gambar 3.11, PCA hanya menangkap karakteristik wajah yang sangat kasar.

### 3.8 FAKTORISASI MATRIKS NON-NEGATIF (NMF)

Faktorisasi matriks non-negatif adalah algoritma pembelajaran tanpa pengawasan lainnya yang bertujuan untuk mengekstrak fitur yang berguna. Ia bekerja mirip dengan PCA dan juga dapat digunakan untuk pengurangan dimensi. Seperti pada PCA, kami mencoba untuk menulis setiap titik data sebagai jumlah bobot dari beberapa komponen, seperti yang diilustrasikan pada Gambar 3.10. Tetapi jika di PCA kami menginginkan komponen yang ortogonal dan menjelaskan sebanyak mungkin varians data, di NMF, kami ingin komponen dan koefisiennya non-negatif; yaitu, kita ingin komponen dan koefisiennya lebih besar dari atau sama dengan nol. Akibatnya, metode ini hanya dapat diterapkan pada data di mana setiap fitur non-negatif, karena jumlah non-negatif dari komponen non-negatif tidak dapat menjadi negatif.

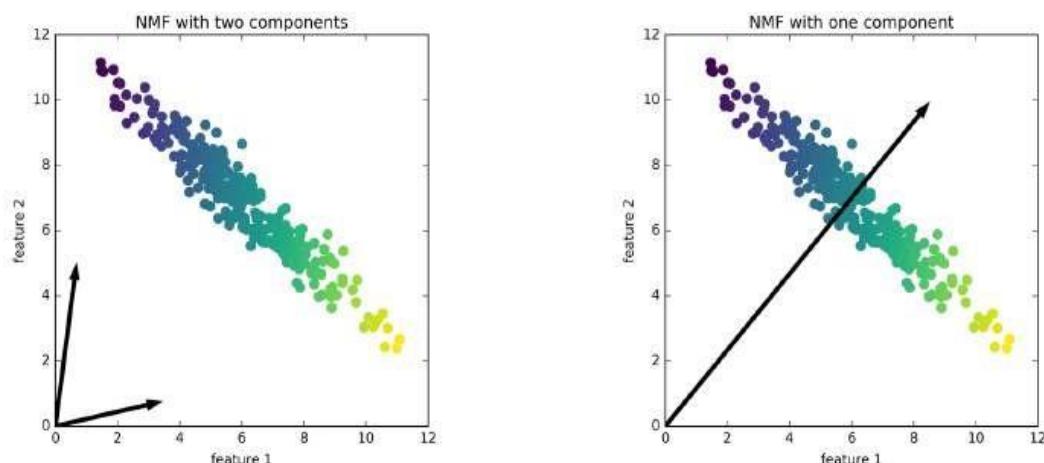
Proses penguraian data menjadi jumlah berbobot non-negatif sangat membantu untuk data yang dibuat sebagai tambahan (atau overlay) dari beberapa sumber independen, seperti trek audio dari beberapa orang yang berbicara, atau musik dengan banyak instrumen. Dalam situasi ini, NMF dapat mengidentifikasi komponen asli yang membentuk data gabungan. Secara keseluruhan, NMF mengarah ke komponen yang lebih dapat diinterpretasikan daripada PCA, karena komponen dan koefisien negatif dapat menyebabkan efek pembatalan yang sulit diinterpretasikan. Eigenfaces pada Gambar 3.9, misalnya, mengandung bagian positif dan negatif, dan seperti yang kami sebutkan dalam deskripsi PCA, tandanya sebenarnya arbitrer. Sebelum kita menerapkan NMF ke dataset wajah, mari kita tinjau kembali data sintetisnya.

#### Menerapkan NMF ke data sintetis

Berbeda dengan saat menggunakan PCA, kita perlu memastikan bahwa data kita positif NMF untuk dapat beroperasi pada data tersebut. Ini berarti di mana letak data relatif terhadap asal  $(0, 0)$  sebenarnya penting untuk NMF. Oleh karena itu, Anda dapat memikirkan komponen non-negatif yang diekstraksi sebagai arah dari  $(0, 0)$  menuju data. Contoh berikut (Gambar 3.13) menunjukkan hasil NMF pada data mainan dua dimensi:

In[34]:

```
mglearn.plots.plot_nmf_illustration()
```



**Gambar 3.13** Komponen yang ditemukan dengan faktorisasi matriks non-negatif dengan dua komponen (kiri) dan satu komponen (kanan)

Untuk NMF dengan dua komponen, seperti yang ditunjukkan di sebelah kiri, jelas bahwa semua titik dalam data dapat ditulis sebagai kombinasi positif dari dua komponen. Jika ada cukup komponen untuk merekonstruksi data dengan sempurna (sebanyak komponen yang ada), algoritme akan memilih arah yang mengarah ke ekstrem data.

Jika kita hanya menggunakan satu komponen, NMF membuat komponen yang mengarah ke mean, karena menunjuk ke sana paling baik menjelaskan data. Anda dapat melihat bahwa berbeda dengan PCA, mengurangi jumlah komponen tidak hanya menghilangkan beberapa arah, tetapi juga menciptakan kumpulan komponen yang sama sekali berbeda! Komponen di NMF juga tidak diurutkan dengan cara tertentu, jadi tidak ada "komponen non-negatif pertama": semua komponen memainkan peran yang sama.

NMF menggunakan inisialisasi acak, yang mungkin menghasilkan hasil yang berbeda tergantung pada benih acak. Dalam kasus yang relatif sederhana seperti data sintetik dengan dua komponen, di mana semua data dapat dijelaskan dengan sempurna, keacakan memiliki sedikit efek (meskipun mungkin mengubah urutan atau skala komponen). Dalam situasi yang lebih kompleks, mungkin ada perubahan yang lebih drastis.

### Menerapkan NMF ke gambar wajah

Sekarang, mari kita terapkan NMF ke Labeled Faces in the Wild dataset yang kita gunakan sebelumnya. Parameter utama NMF adalah berapa banyak komponen yang ingin kita ekstrak. Biasanya ini lebih rendah dari jumlah fitur input (jika tidak, data dapat dijelaskan dengan membuat setiap piksel menjadi komponen terpisah).

Pertama, mari kita periksa bagaimana jumlah komponen memengaruhi seberapa baik data dapat direkonstruksi menggunakan NMF (Gambar 3.14):

In[35]:

```
mglearn.plots.plot_nmf_faces(X_train, X_test, image_shape)
```



**Gambar 3.14** Merekonstruksi tiga gambar wajah menggunakan semakin banyak komponen yang ditemukan oleh NMF

Kualitas data yang diubah kembali mirip dengan saat menggunakan PCA, tetapi sedikit lebih buruk. Hal ini diharapkan, karena PCA menemukan arah optimal dalam hal rekonstruksi. NMF biasanya tidak digunakan karena kemampuannya untuk merekonstruksi atau mengkodekan data, melainkan untuk menemukan pola yang menarik di dalam data.

Sebagai pandangan pertama ke dalam data, mari kita coba mengekstrak hanya beberapa komponen (misalnya, 15). Gambar 3.15 menunjukkan hasil:

**In[36]:**

```
from sklearn.decomposition import NMF
nmf = NMF(n_components=15, random_state=0)
nmf.fit(X_train)
X_train_nmf = nmf.transform(X_train)
X_test_nmf = nmf.transform(X_test)

fix, axes = plt.subplots(3, 5, figsize=(15, 12),
                       subplot_kw={'xticks': (), 'yticks': ()})
for i, (component, ax) in enumerate(zip(nmf.components_, axes.ravel())):
    ax.imshow(component.reshape(image_shape))
    ax.set_title("{} component".format(i))
```



**Gambar 3.15** Komponen yang ditemukan oleh NMF pada dataset wajah saat menggunakan 15 komponen

Semua komponen ini positif, dan lebih menyerupai prototipe wajah daripada komponen yang ditunjukkan untuk PCA pada Gambar 3.9. Sebagai contoh, dapat dilihat dengan jelas bahwa komponen 3 menunjukkan wajah yang agak diputar ke kanan, sedangkan

komponen 7 menunjukkan wajah yang agak diputar ke kiri. Mari kita lihat gambar di mana komponen ini sangat kuat, ditunjukkan pada Gambar 3.16 dan 3.17:

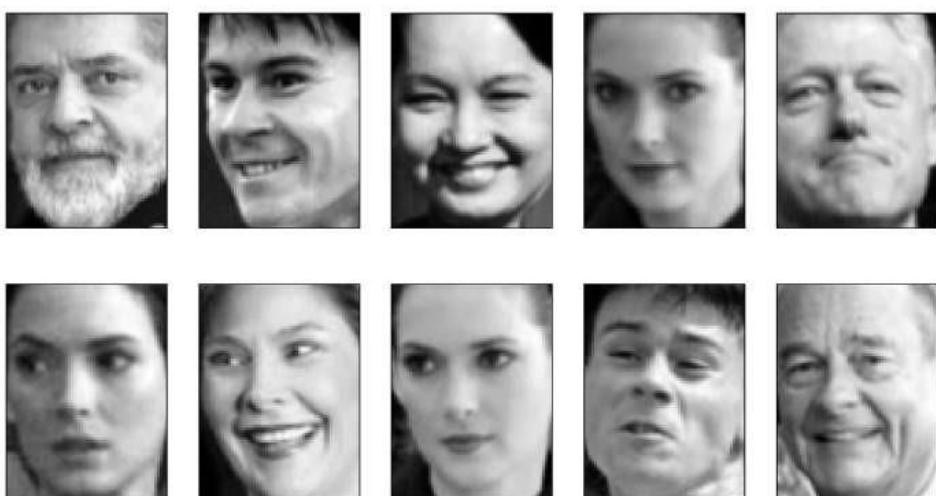
**In[37]:**

```
compn = 3
# sort by 3rd component, plot first 10 images
inds = np.argsort(X_train_nmf[:, compn])[::-1]
fig, axes = plt.subplots(2, 5, figsize=(15, 8),
                        subplot_kw={'xticks': (), 'yticks': ()})
for i, (ind, ax) in enumerate(zip(inds, axes.ravel())):
    ax.imshow(X_train[ind].reshape(image_shape))

compn = 7
# sort by 7th component, plot first 10 images
inds = np.argsort(X_train_nmf[:, compn])[::-1]
fig, axes = plt.subplots(2, 5, figsize=(15, 8),
                        subplot_kw={'xticks': (), 'yticks': ()})
for i, (ind, ax) in enumerate(zip(inds, axes.ravel())):
    ax.imshow(X_train[ind].reshape(image_shape))
```



**Gambar 3.16** Wajah yang memiliki koefisien besar untuk komponen 3



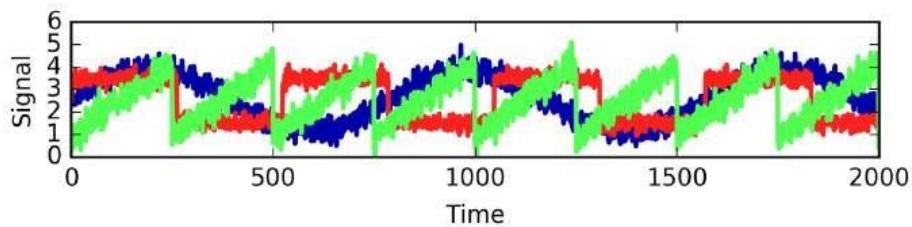
**Gambar 3.17** Wajah yang memiliki koefisien besar untuk komponen 7

Seperti yang diharapkan, wajah yang memiliki koefisien tinggi untuk komponen 3 adalah wajah yang menghadap ke kanan (Gambar 3.16), sedangkan wajah dengan koefisien tinggi untuk komponen 7 menghadap ke kiri (Gambar 3.17). Seperti disebutkan sebelumnya, mengekstraksi pola seperti ini paling cocok untuk data dengan struktur aditif, termasuk audio, ekspresi seni, dan data teks. Mari kita telusuri satu contoh pada data sintetis untuk melihat seperti apa tampilannya.

Katakanlah kita tertarik pada sinyal yang merupakan kombinasi dari tiga sumber yang berbeda (Gambar 3.18):

**In[38]:**

```
S = mglearn.datasets.make_signals()
plt.figure(figsize=(6, 1))
plt.plot(S, '-')
plt.xlabel("Time")
plt.ylabel("Signal")
```



**Gambar 3.18** Sumber sinyal asli

Sayangnya kami tidak dapat mengamati sinyal asli, tetapi hanya campuran aditif dari ketiganya. Kami ingin memulihkan dekomposisi sinyal campuran menjadi komponen asli. Kami berasumsi bahwa kami memiliki banyak cara berbeda untuk mengamati campuran (katakanlah 100 perangkat pengukuran), yang masing-masing memberi kami serangkaian pengukuran:

**In[39]:**

```
# mix data into a 100-dimensional state
A = np.random.RandomState(0).uniform(size=(100, 3))
X = np.dot(S, A.T)
print("Shape of measurements: {}".format(X.shape))
```

**Out[39]:**

```
Shape of measurements: (2000, 100)
```

Kita dapat menggunakan NMF untuk memulihkan tiga sinyal:

**In[40]:**

```
nmf = NMF(n_components=3, random_state=42)
S_ = nmf.fit_transform(X)
print("Recovered signal shape: {}".format(S_.shape))
```

**Out[40]:**

```
Recovered signal shape: (2000, 3)
```

Sebagai perbandingan, kami juga menerapkan PCA:

In[41]:

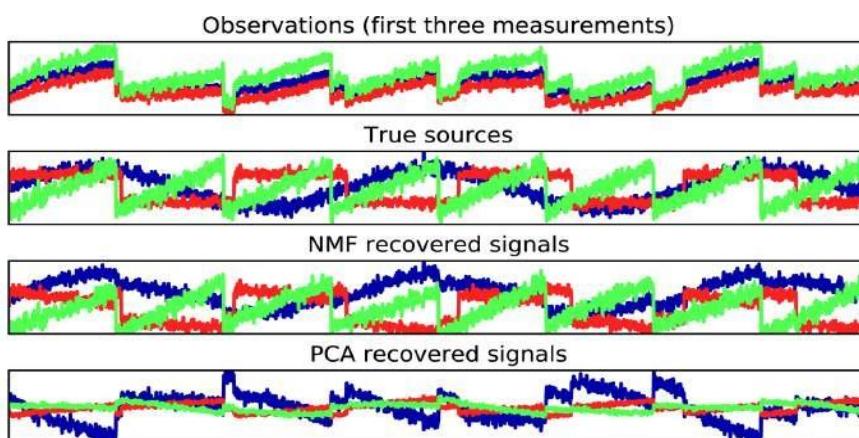
```
pca = PCA(n_components=3)
H = pca.fit_transform(X)
```

In[42]:

```
models = [X, S, S_, H]
names = ['Observations (first three measurements)',
         'True sources',
         'NMF recovered signals',
         'PCA recovered signals']

fig, axes = plt.subplots(4, figsize=(8, 4), gridspec_kw={'hspace': .5},
                       subplot_kw={'xticks': (), 'yticks': ()})

for model, name, ax in zip(models, names, axes):
    ax.set_title(name)
    ax.plot(model[:, :3], '-')
```



Gambar 3.19 Memulihkan sumber campuran menggunakan NMF dan PCA

Angka tersebut mencakup 3 dari 100 pengukuran dari X untuk referensi. Seperti yang Anda lihat, NMF melakukan pekerjaan yang wajar untuk menemukan sumber asli, sementara PCA gagal dan menggunakan komponen pertama untuk menjelaskan sebagian besar variasi dalam data. Perlu diingat bahwa komponen yang diproduksi oleh NMF tidak memiliki urutan alami. Dalam contoh ini, urutan komponen NMF sama dengan sinyal aslinya (lihat bayangan dari tiga kurva), tetapi ini murni kebetulan.

Ada banyak algoritme lain yang dapat digunakan untuk menguraikan setiap titik data menjadi jumlah tertimbang dari sekumpulan komponen tetap, seperti yang dilakukan PCA dan NMF. Membahas semuanya berada di luar cakupan buku ini, dan menjelaskan batasan yang dibuat pada komponen dan koefisien sering kali melibatkan teori probabilitas. Jika Anda tertarik dengan ekstraksi pola semacam ini, kami sarankan Anda mempelajari bagian panduan pengguna sci kit\_learn tentang analisis komponen independen (ICA), analisis faktor (FA), dan pengkodean jarang (pembelajaran kamus), yang semuanya Anda dapat menemukan di halaman tentang metode dekomposisi.

### 3.9 PEMBELAJARAN BERJENIS DENGAN T-SNE

Meskipun PCA sering kali merupakan pendekatan pertama yang baik untuk mengubah data Anda sehingga Anda mungkin dapat memvisualisasikannya menggunakan plot sebar, sifat metode (menerapkan rotasi dan kemudian menurunkan arah) membatasi kegunaannya, seperti yang kita lihat dengan pencar. plot dari dataset Wajah Berlabel di Wild. Ada kelas algoritme untuk visualisasi yang disebut algoritme pembelajaran manifold yang memungkinkan pemetaan yang jauh lebih kompleks, dan seringkali memberikan visualisasi yang lebih baik. Salah satu yang sangat berguna adalah algoritma t-SNE.

Algoritma pembelajaran manifold terutama ditujukan untuk visualisasi, sehingga jarang digunakan untuk menghasilkan lebih dari dua fitur baru. Beberapa dari mereka, termasuk t-SNE, menghitung representasi baru dari data pelatihan, tetapi tidak mengizinkan transformasi data baru. Ini berarti algoritme ini tidak dapat diterapkan ke set pengujian: alih-alih, algoritma ini hanya dapat mengubah data yang dilatih untuknya.

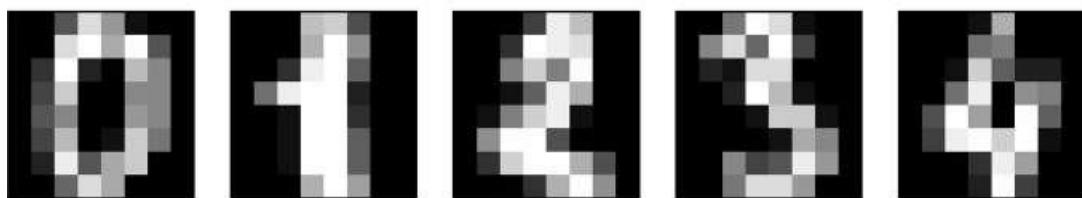
Pembelajaran berjenis dapat berguna untuk analisis data eksplorasi, tetapi jarang digunakan jika tujuan akhir adalah pembelajaran yang diawasi. Ide dibalik t-SNE adalah untuk menemukan representasi dua dimensi dari data yang menjaga jarak antar titik sebaik mungkin. t-SNE dimulai dengan representasi dua dimensi acak untuk setiap titik data, dan kemudian mencoba untuk membuat titik-titik yang dekat di ruang fitur asli lebih dekat, dan titik-titik yang berjauhan di ruang fitur asli lebih jauh. t-SNE lebih menekankan pada titik-titik yang dekat, daripada menjaga jarak antara titik-titik yang berjauhan. Dengan kata lain, ia mencoba untuk menyimpan informasi yang menunjukkan titik mana yang bertetangga satu sama lain.

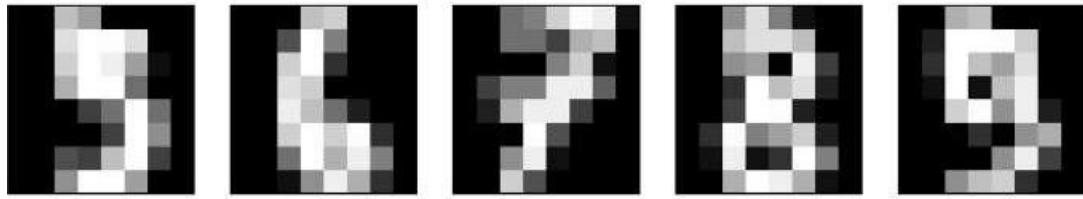
Kami akan menerapkan algoritma pembelajaran berjenis t-SNE pada dataset digit tulisan tangan yang termasuk dalam scikit-learn.2 Setiap titik data dalam dataset ini adalah gambar skala abu-abu 8x8 dari digit tulisan tangan antara 0 dan 1. Gambar 3.20 menunjukkan contoh gambar untuk setiap kelas:

**In[43]:**

```
from sklearn.datasets import load_digits
digits = load_digits()

fig, axes = plt.subplots(2, 5, figsize=(10, 5),
                       subplot_kw={'xticks':(), 'yticks': ()})
for ax, img in zip(axes.ravel(), digits.images):
    ax.imshow(img)
```





**Gambar 3.20** Contoh gambar dari kumpulan data digit

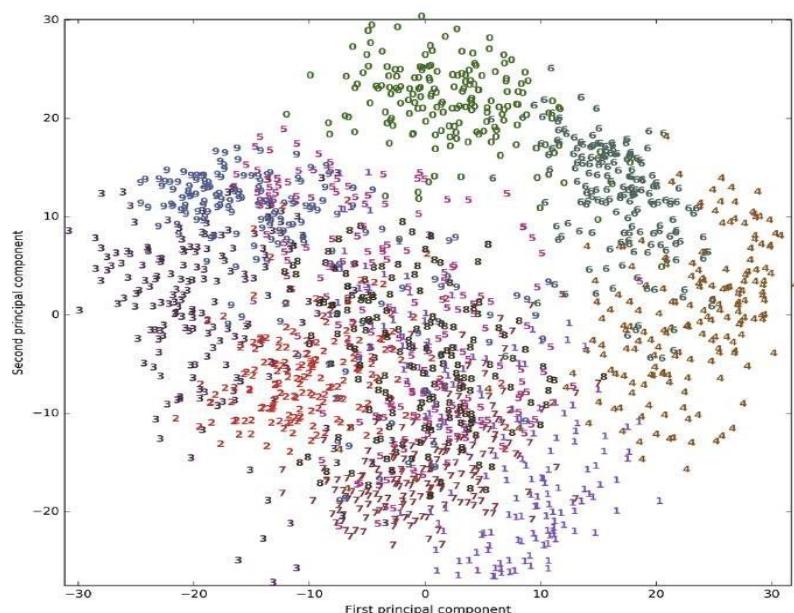
Mari gunakan PCA untuk memvisualisasikan data yang direduksi menjadi dua dimensi. Kami memplot dua komponen utama pertama, dan mewarnai setiap titik berdasarkan kelasnya (lihat Gambar 3.21):

In[44]:

```
# build a PCA model
pca = PCA(n_components=2)
pca.fit(digits.data)

# transform the digits data onto the first two principal components
digits_pca = pca.transform(digits.data)
colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525",
          "#A83683", "#4E655E", "#853541", "#3A3120", "#535D8E"]
plt.figure(figsize=(10, 10))
plt.xlim(digits_pca[:, 0].min(), digits_pca[:, 0].max())
plt.ylim(digits_pca[:, 1].min(), digits_pca[:, 1].max())
for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
              color = colors[digits.target[i]],
              fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```

Di sini, kami benar-benar menggunakan kelas digit sebenarnya sebagai mesin terbang, untuk menunjukkan kelas mana. Angka nol, enam, dan empat relatif terpisah dengan baik menggunakan dua komponen utama pertama, meskipun masih tumpang tindih. Sebagian besar digit lainnya tumpang tindih secara signifikan.



**Gambar 3.21** Plot pencar dari kumpulan data digit menggunakan dua komponen utama pertama

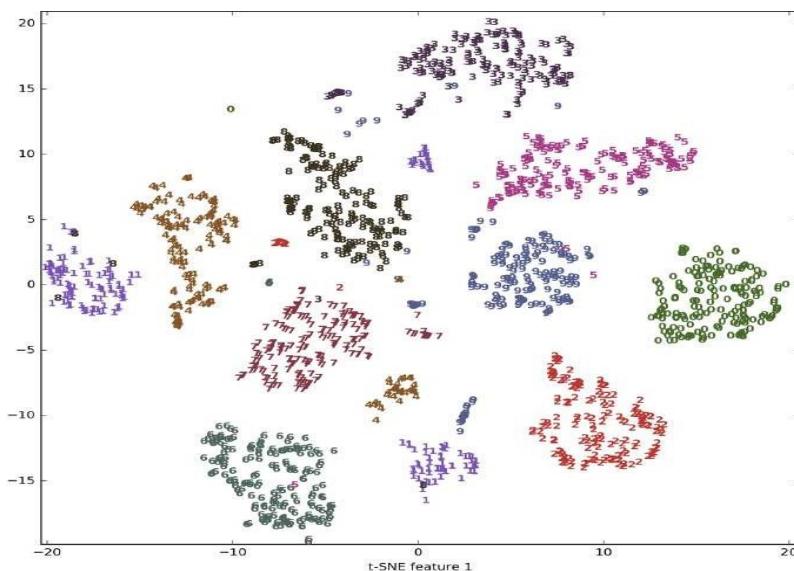
Mari kita terapkan t-SNE ke dataset yang sama, dan bandingkan hasilnya. Karena t-SNE tidak mendukung transformasi data baru, kelas TSNE tidak memiliki metode transformasi. Sebagai gantinya, kita dapat memanggil metode fit\_transform, yang akan membangun model dan segera mengembalikan data yang diubah (lihat Gambar 3.22):

In[45]:

```
from sklearn.manifold import TSNE
tsne = TSNE(random_state=42)
# use fit_transform instead of fit, as TSNE has no transform method
digits_tsne = tsne.fit_transform(digits.data)
```

In[46]:

```
plt.figure(figsize=(10, 10))
plt.xlim(digits_tsne[:, 0].min(), digits_tsne[:, 0].max() + 1)
plt.ylim(digits_tsne[:, 1].min(), digits_tsne[:, 1].max() + 1)
for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_tsne[i, 0], digits_tsne[i, 1], str(digits.target[i]),
              color = colors[digits.target[i]],
              fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("t-SNE feature 0")
plt.ylabel("t-SNE feature 1")
```



**Gambar 3.22** Scatter plot dari dataset digit menggunakan dua komponen yang ditemukan oleh t-SNE

Hasil t-SNE cukup luar biasa. Semua kelas dipisahkan dengan cukup jelas. Satu dan sembilan agak terpisah, tetapi sebagian besar kelas membentuk satu kelompok padat. Perlu diingat bahwa metode ini tidak memiliki pengetahuan tentang label kelas: metode ini sama

sekali tidak diawasi. Namun, ia dapat menemukan representasi data dalam dua dimensi yang secara jelas memisahkan kelas, hanya berdasarkan seberapa dekat titik dalam ruang aslinya.

Algoritme t-SNE memiliki beberapa parameter penyetelan, meskipun sering kali bekerja dengan baik dengan pengaturan default. Anda dapat mencoba bermain dengan kebingungan dan awal\_berlebihan, tetapi efeknya biasanya kecil.

### 3.10 PEKELOMPOKAN

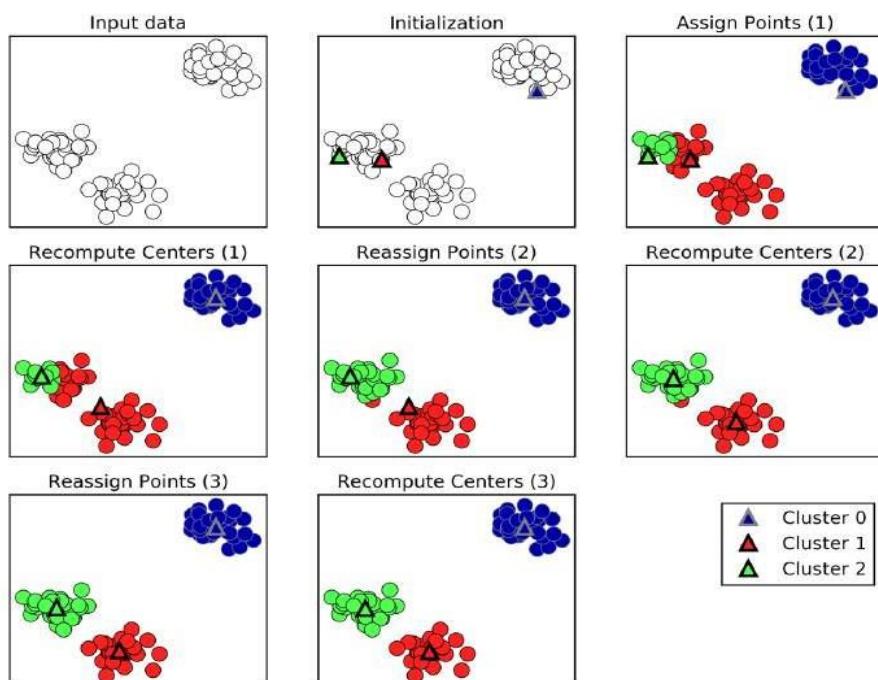
Seperti yang kami jelaskan sebelumnya, clustering adalah tugas mempartisi dataset ke dalam grup, yang disebut cluster. Tujuannya adalah untuk membagi data sedemikian rupa sehingga titik-titik dalam satu cluster sangat mirip dan titik-titik dalam cluster yang berbeda berbeda. Serupa dengan algoritma klasifikasi, algoritma pengelompokan menetapkan (atau memprediksi) nomor untuk setiap titik data, menunjukkan cluster mana yang dimiliki titik tertentu.

### 3.11 K-MEANS CLUSTERING

k-means clustering adalah salah satu algoritma clustering yang paling sederhana dan paling umum digunakan. Ini mencoba untuk menemukan pusat cluster yang mewakili wilayah tertentu dari data. Algoritme bergantian antara dua langkah: menetapkan setiap titik data ke pusat cluster terdekat, dan kemudian menetapkan setiap pusat cluster sebagai rata-rata dari titik data yang ditugaskan padanya. Algoritme selesai ketika penugasan instance ke cluster tidak lagi berubah. Contoh berikut (Gambar 3.23) mengilustrasikan algoritme pada kumpulan data sintetis:

In[47]:

```
mglearn.plots.plot_kmeans_algorithm()
```



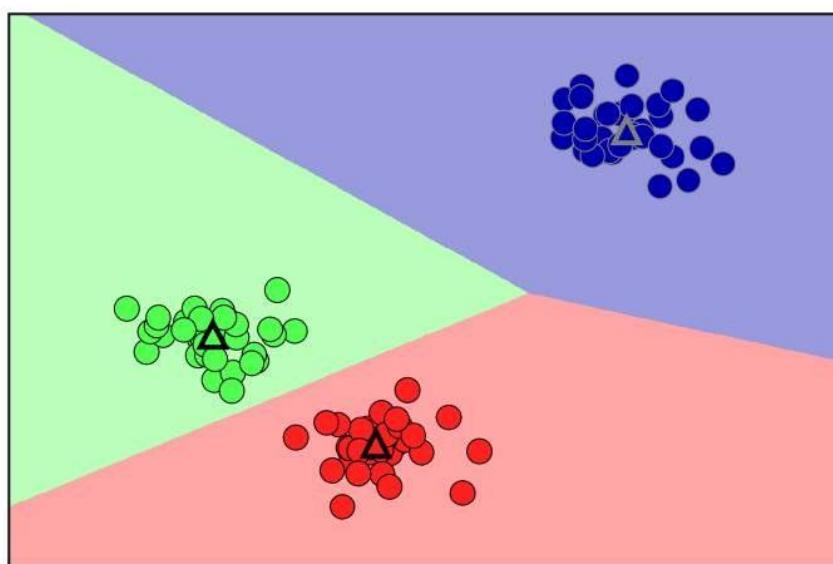
Gambar 3.23 Input data dan tiga langkah dari algoritma k-means

Pusat cluster ditampilkan sebagai segitiga, sedangkan titik data ditampilkan sebagai lingkaran. Warna menunjukkan keanggotaan cluster. Kami menetapkan bahwa kami sedang mencari tiga cluster, sehingga algoritme diinisialisasi dengan mendeklarasikan tiga titik data secara acak sebagai pusat cluster (lihat “Inisialisasi”). Kemudian algoritma iteratif dimulai. Pertama, setiap titik data ditugaskan ke pusat cluster yang paling dekat dengannya (lihat “Menetapkan Poin (1)”). Selanjutnya, pusat cluster diperbarui menjadi rata-rata dari poin yang ditetapkan (lihat “Pusat Hitung Ulang (1)”). Kemudian proses diulang dua kali lagi. Setelah iterasi ketiga, penetapan titik ke pusat cluster tetap tidak berubah, sehingga algoritma berhenti.

Mengingat titik data baru, k-means akan menetapkan masing-masing ke pusat cluster terdekat. Contoh berikutnya (Gambar 3.24) menunjukkan batas-batas pusat cluster yang dipelajari pada Gambar 3.23:

**In[48]:**

```
mglearn.plots.plot_kmeans_boundaries()
```



**Gambar 3.24** Pusat cluster dan batas cluster ditemukan oleh algoritma k-means

Menerapkan k-means dengan scikit-learn cukup mudah. Di sini, kami menerapkannya pada data sintetis yang kami gunakan untuk plot sebelumnya. Kami membuat instance kelas KMeans, dan mengatur jumlah cluster yang kami cari.3 Kemudian kami memanggil metode fit dengan data:

**In[49]:**

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# generate synthetic two-dimensional data
X, y = make_blobs(random_state=1)

# build the clustering model
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

Selama algoritma, setiap titik data pelatihan di X diberi label cluster. Anda dapat menemukan label ini di atribut kmeans.labels\_:

**In[50]:**

```
print("Cluster memberships:\n{}".format(kmeans.labels_))
```

**Out[50]:**

```
Cluster memberships:
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 0 1 2 0 0 1 1 0 1 1 1 0 1 2 0 2
 2 2 0 0 2 1 2 2 0 1 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1
 1 2 0 0 1 2 1 2 2 0 1 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1]
```

Saat kami meminta tiga cluster, cluster diberi nomor 0 hingga 2. Anda juga dapat menetapkan label cluster ke titik baru, menggunakan metode prediksi. Setiap titik baru ditugaskan ke pusat cluster terdekat saat memprediksi, tetapi model yang ada tidak berubah. Menjalankan prediksi pada set pelatihan mengembalikan hasil yang sama seperti labels\_:

**In[51]:**

```
print(kmeans.predict(X))
```

**Out[51]:**

```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2
 2 2 0 0 2 1 2 2 0 1 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1
 1 2 0 0 1 2 1 2 2 0 1 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1]
```

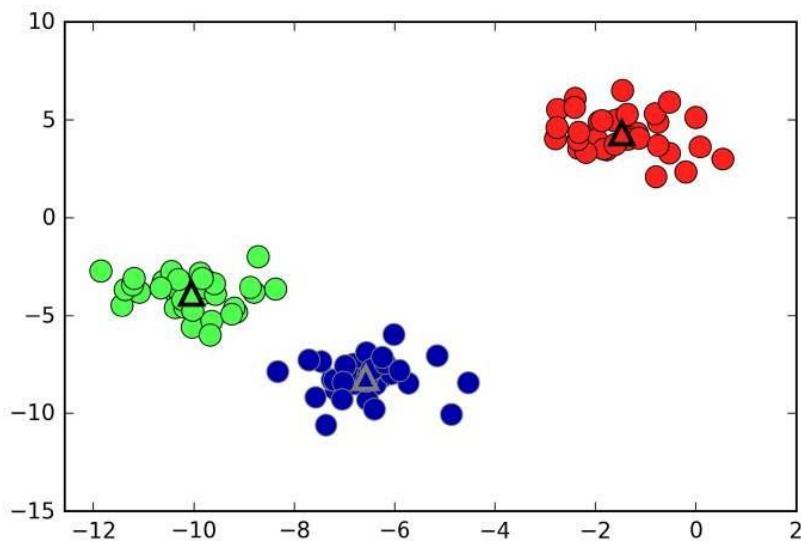
Anda dapat melihat bahwa pengelompokan agak mirip dengan klasifikasi, di mana setiap item mendapat label. Namun, tidak ada kebenaran dasar, dan akibatnya label itu sendiri tidak memiliki makna apriori. Mari kembali ke contoh clustering gambar wajah yang telah kita bahas sebelumnya. Mungkin saja cluster 3 yang ditemukan oleh algoritma hanya berisi wajah teman Anda Bela. Anda hanya dapat mengetahuinya setelah Anda melihat gambar-gambarnya, dan angka 3 adalah sembarang. Satu-satunya informasi yang diberikan algoritme kepada Anda adalah bahwa semua wajah yang diberi label 3 serupa.

Untuk pengelompokan yang baru saja kita hitung pada dataset mainan dua dimensi, itu berarti bahwa kita tidak boleh memberikan signifikansi apa pun pada fakta bahwa satu kelompok diberi label 0 dan yang lain diberi label 1. Menjalankan algoritme lagi mungkin menghasilkan perbedaan penomoran cluster karena sifat inisialisasi yang acak.

Berikut adalah plot data ini lagi (Gambar 3.25). Pusat cluster disimpan di cluster\_centers\_ atribut, dan kami memplotnya sebagai segitiga:

**In[52]:**

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers='o')
mglearn.discrete_scatter(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], [0, 1, 2],
    markers='^', markeredgewidth=2)
```



**Gambar 3.25** Penugasan cluster dan pusat cluster ditemukan oleh k-means dengan tiga cluster

Kami juga dapat menggunakan lebih banyak atau lebih sedikit pusat cluster (Gambar 3.26):

**In[53]:**

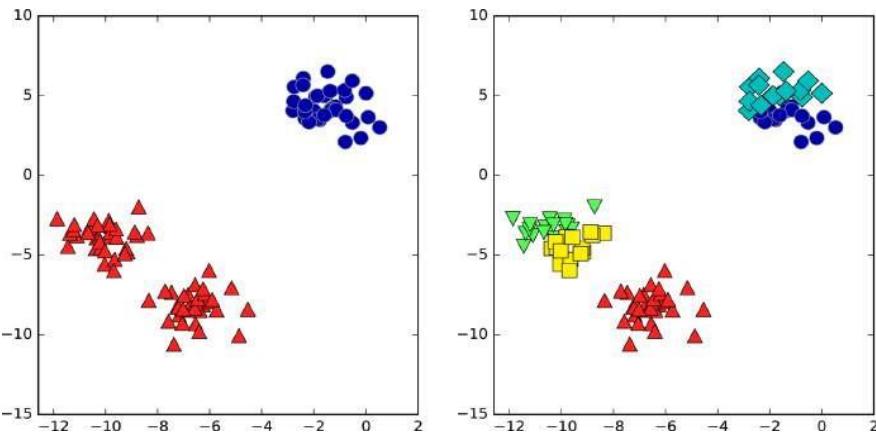
```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# using two cluster centers:
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[0])

# using five cluster centers:
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[1])
```



**Gambar 3.26** Penugasan cluster ditemukan oleh k-means menggunakan dua cluster (kiri) dan lima cluster (kanan)

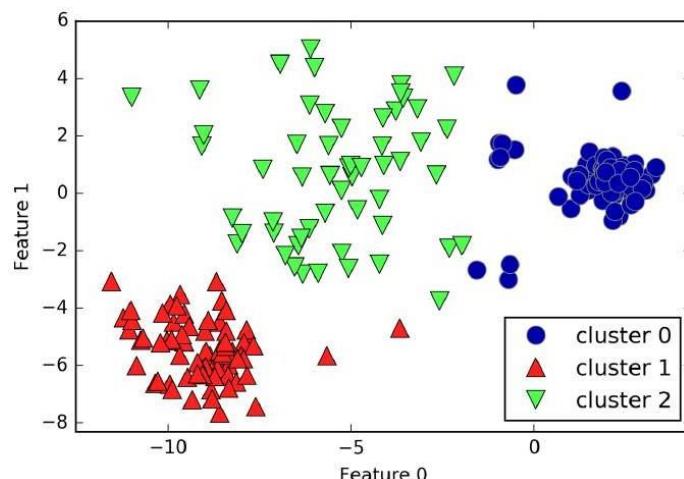
### Kasus kegagalan k-means

Bahkan jika Anda mengetahui jumlah cluster yang "benar" untuk kumpulan data tertentu, k-means mungkin tidak selalu dapat memulihkannya. Setiap cluster hanya ditentukan oleh pusatnya, yang berarti bahwa setiap cluster adalah bentuk cembung. Akibatnya, k-means hanya dapat menangkap bentuk yang relatif sederhana. k-means juga mengasumsikan bahwa semua cluster memiliki "diameter" yang sama dalam beberapa hal; itu selalu menarik batas antar cluster tepat di tengah-tengah antara pusat cluster. Hal itu terkadang dapat menghasilkan hasil yang mengejutkan, seperti yang ditunjukkan pada Gambar 3.27:

**In[54]:**

```
X_varied, y_varied = make_blobs(n_samples=200,
                                 cluster_std=[1.0, 2.5, 0.5],
                                 random_state=170)
y_pred = KMeans(n_clusters=3, random_state=0).fit_predict(X_varied)

mglearn.discrete_scatter(X_varied[:, 0], X_varied[:, 1], y_pred)
plt.legend(["cluster 0", "cluster 1", "cluster 2"], loc='best')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 3.27** Penugasan cluster ditemukan oleh k-means ketika cluster memiliki kepadatan yang berbeda

Orang mungkin mengharapkan wilayah padat di kiri bawah menjadi cluster pertama, wilayah padat di kanan atas menjadi yang kedua, dan wilayah kurang padat di tengah menjadi yang ketiga. Sebaliknya, baik cluster 0 maupun cluster 1 memiliki beberapa titik yang jauh dari semua titik lain dalam cluster ini yang “mencapai” menuju pusat.

k-means juga mengasumsikan bahwa semua arah sama pentingnya untuk setiap cluster. Plot berikut (Gambar 3.28) menunjukkan dataset dua dimensi di mana ada tiga bagian data yang terpisah dengan jelas. Namun, kelompok-kelompok ini membentang ke arah diagonal. Karena k-means hanya mempertimbangkan jarak ke pusat cluster terdekat, k-means tidak dapat menangani jenis data ini:

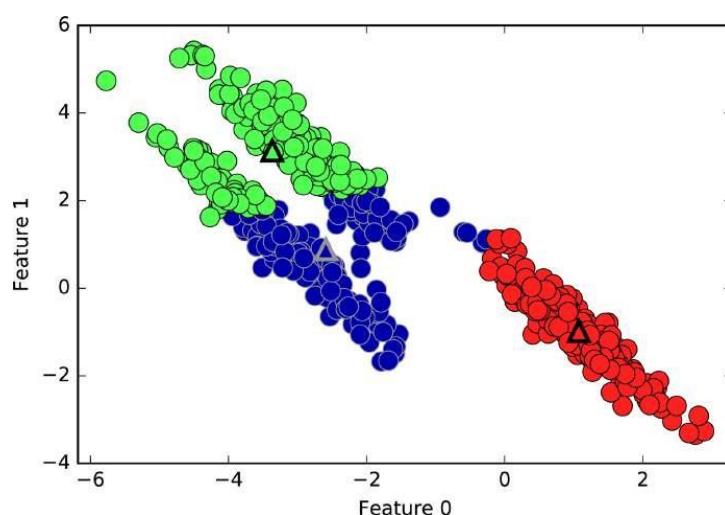
**In[55]:**

```
# generate some random cluster data
X, y = make_blobs(random_state=170, n_samples=600)
rng = np.random.RandomState(74)

# transform the data to be stretched
transformation = rng.normal(size=(2, 2))
X = np.dot(X, transformation)

# cluster the data into three clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# plot the cluster assignments and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm3)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='^', c=[0, 1, 2], s=100, linewidth=2, cmap=mglearn.cm3)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 3.28** k-means gagal mengidentifikasi cluster nonspherical

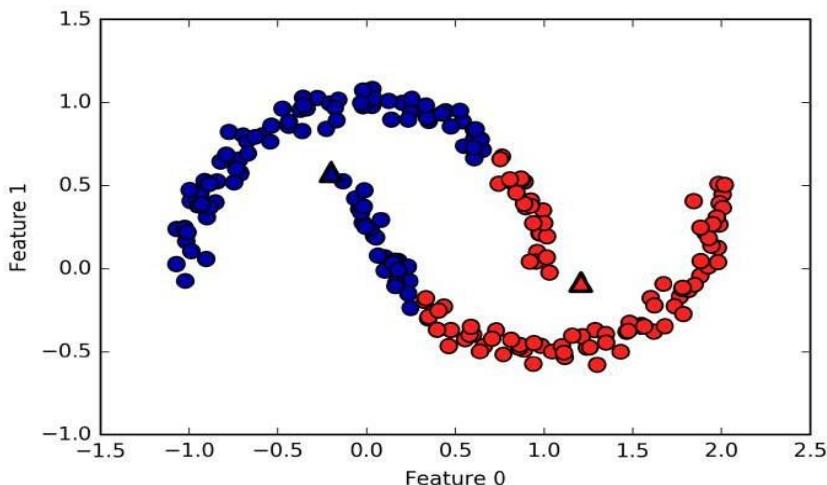
k-means juga berkinerja buruk jika cluster memiliki bentuk yang lebih kompleks, seperti data two\_moons yang kita temui di Bab 2 (lihat Gambar 3.29):

In[56]:

```
# generate synthetic two_moons data (with less noise this time)
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# cluster the data into two clusters
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# plot the cluster assignments and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mlearn.cm2, s=60)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='^', c=[mlearn.cm2(0), mlearn.cm2(1)], s=100, linewidth=2)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 3.29** k-means gagal mengidentifikasi cluster dengan bentuk kompleks

Di sini, kami berharap bahwa algoritma pengelompokan dapat menemukan dua bentuk setengah bulan. Namun, hal ini tidak mungkin dilakukan dengan menggunakan algoritma k-means.

#### Kuantisasi vektor, atau melihat k-means sebagai dekomposisi

Meskipun k-means adalah algoritma pengelompokan, ada persamaan yang menarik antara k-means dan metode dekomposisi seperti PCA dan NMF yang telah kita bahas sebelumnya. Anda mungkin ingat bahwa PCA mencoba menemukan arah varians maksimum dalam data, sementara NMF mencoba menemukan komponen aditif, yang sering kali sesuai dengan "ekstrim" atau "bagian" dari data (lihat Gambar 3.13). Kedua metode mencoba untuk mengekspresikan titik-titik data sebagai jumlah dari beberapa komponen. k-means, di sisi lain, mencoba untuk mewakili setiap titik data menggunakan pusat cluster. Anda dapat menganggapnya sebagai setiap titik yang diwakili hanya menggunakan satu komponen, yang diberikan oleh pusat cluster. Pandangan k-means sebagai metode dekomposisi, di mana setiap titik diwakili menggunakan komponen tunggal, disebut kuantisasi vektor.

Mari kita lakukan perbandingan PCA, NMF, dan k-means secara berdampingan, menunjukkan komponen yang diekstraksi (Gambar 3.30), serta rekonstruksi wajah dari set uji menggunakan 100 komponen (Gambar 3.31). Untuk k-means, rekonstruksi adalah pusat cluster terdekat yang ditemukan pada set pelatihan:

**In[57]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
nmf = NMF(n_components=100, random_state=0)
nmf.fit(X_train)
pca = PCA(n_components=100, random_state=0)
pca.fit(X_train)
kmeans = KMeans(n_clusters=100, random_state=0)
kmeans.fit(X_train)
```

**In[58]:**

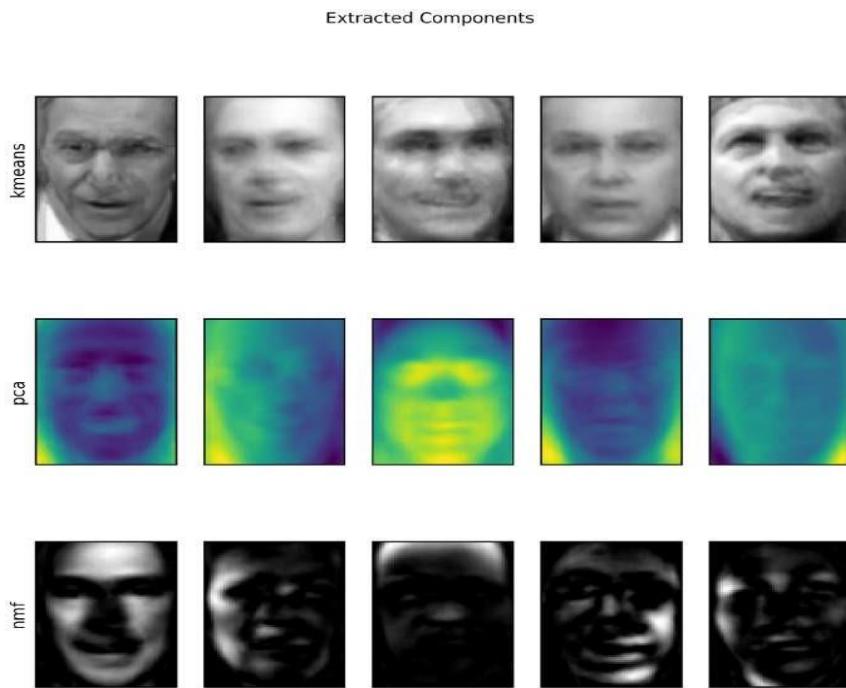
```
fig, axes = plt.subplots(3, 5, figsize=(8, 8),
                       subplot_kw={'xticks': (), 'yticks': ()})
fig.suptitle("Extracted Components")
for ax, comp_kmeans, comp_pca, comp_nmf in zip(
    axes.T, kmeans.cluster_centers_, pca.components_, nmf.components_):
    ax[0].imshow(comp_kmeans.reshape(image_shape))
    ax[1].imshow(comp_pca.reshape(image_shape), cmap='viridis')
    ax[2].imshow(comp_nmf.reshape(image_shape))

axes[0, 0].set_ylabel("kmeans")
axes[1, 0].set_ylabel("pca")
axes[2, 0].set_ylabel("nmf")

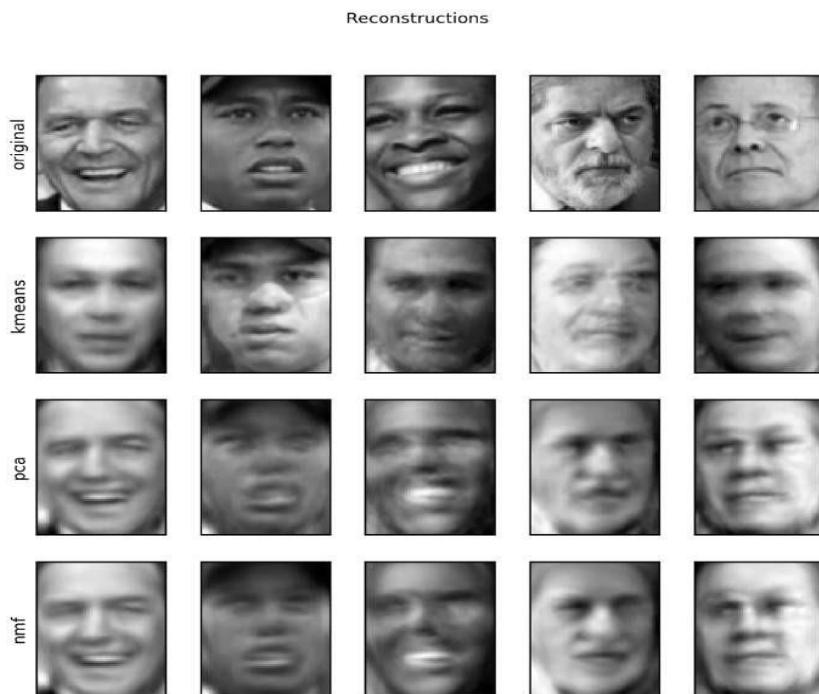
fig, axes = plt.subplots(4, 5, subplot_kw={'xticks': (), 'yticks': ()},
                       figsize=(8, 8))
fig.suptitle("Reconstructions")
for ax, orig, rec_kmeans, rec_pca, rec_nmf in zip(
    axes.T, X_test, X_reconstructed_kmeans, X_reconstructed_pca,
    X_reconstructed_nmf):

    ax[0].imshow(orig.reshape(image_shape))
    ax[1].imshow(rec_kmeans.reshape(image_shape))
    ax[2].imshow(rec_pca.reshape(image_shape))
    ax[3].imshow(rec_nmf.reshape(image_shape))

axes[0, 0].set_ylabel("original")
axes[1, 0].set_ylabel("kmeans")
axes[2, 0].set_ylabel("pca")
axes[3, 0].set_ylabel("nmf")
```



**Gambar 3.30** Membandingkan pusat cluster k-means dengan komponen yang ditemukan oleh PCA dan NMF



**Gambar 3.31** Membandingkan rekonstruksi gambar menggunakan k-means, PCA, dan NMF dengan 100 komponen (atau pusat cluster)—k-means hanya menggunakan satu pusat cluster per gambar

Aspek yang menarik dari kuantisasi vektor menggunakan k-means adalah bahwa kita dapat menggunakan lebih banyak cluster daripada dimensi input untuk mengkodekan data kita. Mari kembali ke data `two_moons`. Menggunakan PCA atau NMF, tidak banyak yang dapat kita lakukan untuk data ini, karena data ini hanya hidup dalam dua dimensi. Menguranginya

menjadi satu dimensi dengan PCA atau NMF akan benar-benar menghancurkan struktur data. Tetapi kita dapat menemukan representasi yang lebih ekspresif dengan k-means, dengan menggunakan lebih banyak pusat cluster (lihat Gambar 3.32):

**In[59]:**

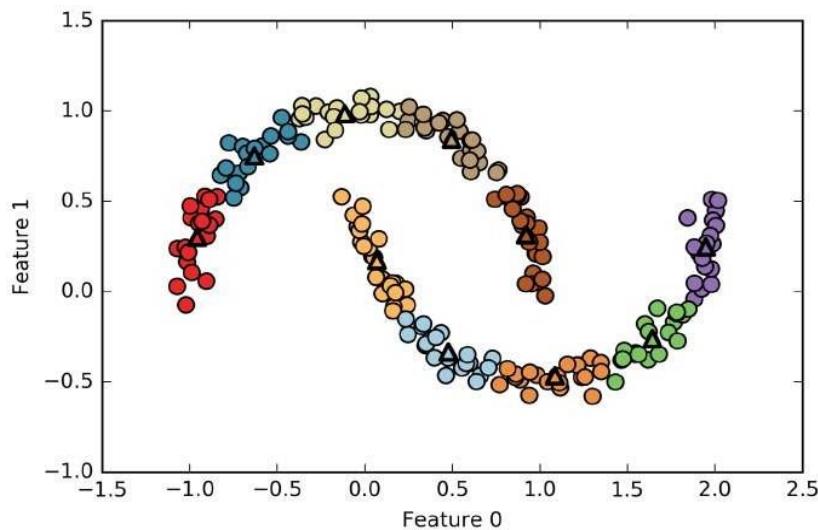
```
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(X)
y_pred = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_pred, s=60, cmap='Paired')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=60,
            marker='^', c=range(kmeans.n_clusters), linewidth=2, cmap='Paired')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
print("Cluster memberships:\n{}".format(y_pred))
```

**Out[59]:**

```
Cluster memberships:
[9 2 5 4 2 7 9 6 9 6 1 0 2 6 1 9 3 0 3 1 7 6 8 6 8 5 2 7 5 8 9 8 6 5 3 7 0
 9 4 5 0 1 3 5 2 8 9 1 5 6 1 0 7 4 6 3 3 6 3 8 0 4 2 9 6 4 8 2 8 4 0 4 0 5
 6 4 5 9 3 0 7 8 0 7 5 8 9 8 0 7 3 9 7 1 7 2 2 0 4 5 6 7 8 9 4 5 4 1 2 3 1
 8 8 4 9 2 3 7 0 9 9 1 5 8 5 1 9 5 6 7 9 1 4 0 6 2 6 4 7 9 5 5 3 8 1 9 5 6
 3 5 0 2 9 3 0 8 6 0 3 3 5 6 3 2 0 2 3 0 2 6 3 4 4 1 5 6 7 1 1 3 2 4 7 2 7
 3 8 6 4 1 4 3 9 9 5 1 7 5 8 2]
```



**Gambar 3.32** Menggunakan banyak k-means clusters untuk menutupi variasi dalam kumpulan data yang kompleks

Kami menggunakan 10 pusat cluster, yang berarti setiap titik sekarang diberi nomor antara 0 dan 9. Kami dapat melihat ini sebagai data yang diwakili menggunakan 10 komponen (yaitu, kami memiliki 10 fitur baru), dengan semua fitur menjadi 0, terpisah dari salah satu yang mewakili pusat cluster titik ditugaskan. Dengan menggunakan representasi 10-dimensi ini, sekarang dimungkinkan untuk memisahkan dua bentuk setengah bulan menggunakan model linier, yang tidak mungkin dilakukan dengan menggunakan dua fitur asli. Dimungkinkan juga untuk mendapatkan representasi data yang lebih ekspresif dengan menggunakan jarak

ke masing-masing pusat cluster sebagai fitur. Ini dapat dicapai dengan menggunakan metode transformasi kmeans:

**In[60]:**

```
distance_features = kmeans.transform(X)
print("Distance feature shape: {}".format(distance_features.shape))
print("Distance features:\n{}".format(distance_features))
```

**Out[60]:**

```
Distance feature shape: (200, 10)
Distance features:
[[ 0.922  1.466  1.14    ...,  1.166  1.039  0.233]
 [ 1.142  2.517  0.12    ...,  0.707  2.204  0.983]
 [ 0.788  0.774  1.749  ...,  1.971  0.716  0.944]
 ...,
 [ 0.446  1.106  1.49    ...,  1.791  1.032  0.812]
 [ 1.39   0.798  1.981  ...,  1.978  0.239  1.058]
 [ 1.149  2.454  0.045  ...,  0.572  2.113  0.882]]
```

k-means adalah algoritma yang sangat populer untuk pengelompokan, tidak hanya karena relatif mudah dipahami dan diimplementasikan, tetapi juga karena berjalan relatif cepat. k-means menskalakan dengan mudah ke kumpulan data besar, dan scikit-learn bahkan menyertakan varian yang lebih skalabel di kelas MiniBatchKMeans, yang dapat menangani kumpulan data yang sangat besar.

Salah satu kelemahan k-means adalah bergantung pada inisialisasi acak, yang berarti hasil dari algoritma bergantung pada seed acak. Secara default, scikit-learning menjalankan algoritma 10 kali dengan 10 inisialisasi acak yang berbeda, dan mengembalikan hasil terbaik.<sup>4</sup> Kelemahan lebih lanjut dari k-means adalah asumsi yang relatif terbatas yang dibuat pada bentuk cluster, dan persyaratan untuk menentukan jumlah ber dari cluster yang Anda cari (yang mungkin tidak diketahui dalam aplikasi dunia nyata).

Selanjutnya, kita akan melihat dua algoritma pengelompokan lagi yang meningkatkan sifat-sifat ini dalam beberapa cara.

### 3.12 PENGELOMPOKAN AGGLOMERATIVE

Pengelompokan aglomeratif mengacu pada kumpulan algoritma pengelompokan yang semuanya dibangun berdasarkan prinsip yang sama: algoritma dimulai dengan mendeklarasikan setiap titik klasternya sendiri, dan kemudian menggabungkan dua klaster yang paling mirip sampai beberapa kriteria penghentian terpenuhi. Kriteria penghentian yang diterapkan dalam scikit-learn adalah jumlah cluster, sehingga cluster yang sama digabung sampai hanya jumlah cluster yang ditentukan yang tersisa. Ada beberapa kriteria keterkaitan yang menentukan bagaimana tepatnya "cluster yang paling mirip" diukur. Ukuran ini selalu didefinisikan antara dua cluster yang ada.

Tiga pilihan berikut diimplementasikan dalam scikit-learn:

*ward*

Pilihan default, ward memilih dua cluster untuk digabungkan sedemikian rupa sehingga varians

dalam semua cluster meningkat paling sedikit. Hal ini sering menyebabkan cluster yang berukuran relatif sama.

*rata-rata*

average linkage menggabungkan dua cluster yang memiliki jarak rata-rata terkecil antara semua titiknya.

*complete*

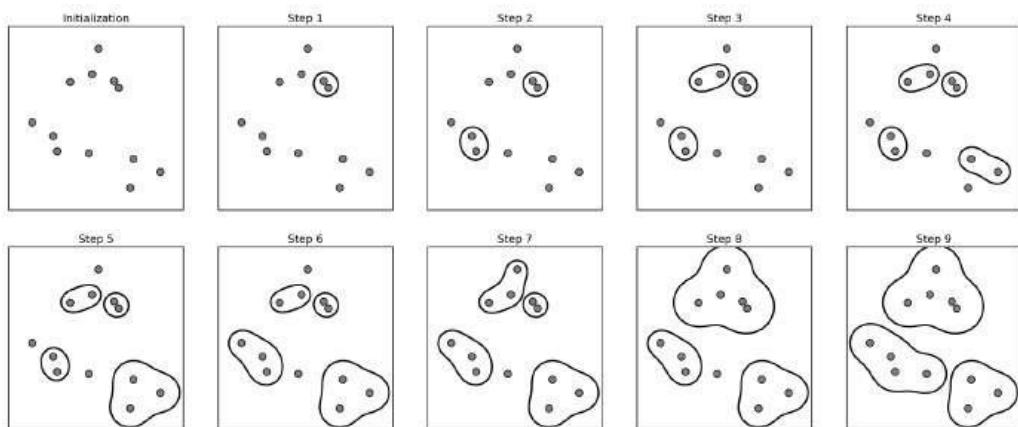
complete linkage (juga dikenal sebagai linkage maksimum) menggabungkan dua cluster yang memiliki jarak maksimum terkecil antara titik-titiknya.

Word bekerja pada sebagian besar kumpulan data, dan kami akan menggunakan dalam contoh kami. Jika cluster memiliki jumlah anggota yang sangat berbeda (jika salah satu lebih besar dari yang lain, misalnya), rata-rata atau lengkap mungkin bekerja lebih baik.

Plot berikut (Gambar 3.33) menggambarkan perkembangan pengelompokan aglomeratif pada dataset dua dimensi, mencari tiga cluster:

In[61]:

```
mlearn.plots.plot_agglomerative_algorithm()
```



**Gambar 3.33** Pengelompokan aglomerat secara iteratif bergabung dengan dua cluster terdekat

Awalnya, setiap titik adalah clusternya sendiri. Kemudian, pada setiap langkah, dua cluster yang paling dekat digabungkan. Dalam empat langkah pertama, dua kluster titik tunggal diambil dan ini digabungkan menjadi kluster dua titik. Pada langkah 5, salah satu cluster dua titik diperluas ke titik ketiga, dan seterusnya. Pada langkah 9, hanya ada tiga cluster yang tersisa. Seperti yang kami tentukan bahwa kami sedang mencari tiga cluster, algoritma kemudian berhenti.

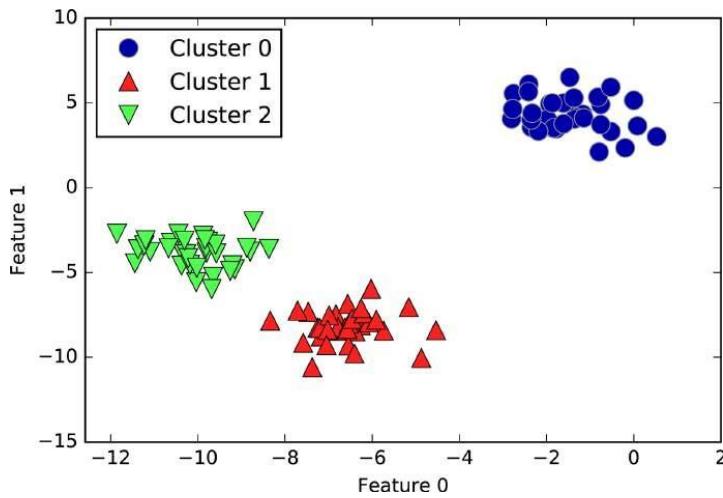
Mari kita lihat bagaimana kinerja agglomerative clustering pada data tiga cluster sederhana yang kita gunakan di sini. Karena cara kerja algoritme, pengelompokan aglomerasi tidak dapat membuat prediksi untuk titik data baru. Oleh karena itu, Agglomerative Clustering tidak memiliki metode prediksi. Untuk membangun model dan mendapatkan anggota cluster pada set pelatihan, gunakan metode fit\_predict sebagai gantinya. Hasilnya ditunjukkan pada Gambar 3.34:

**In[62]:**

```
from sklearn.cluster import AgglomerativeClustering
X, y = make_blobs(random_state=1)

agg = AgglomerativeClustering(n_clusters=3)
assignment = agg.fit_predict(X)

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```



**Gambar 3.34** Penugasan cluster menggunakan clustering aglomeratif dengan tiga cluster

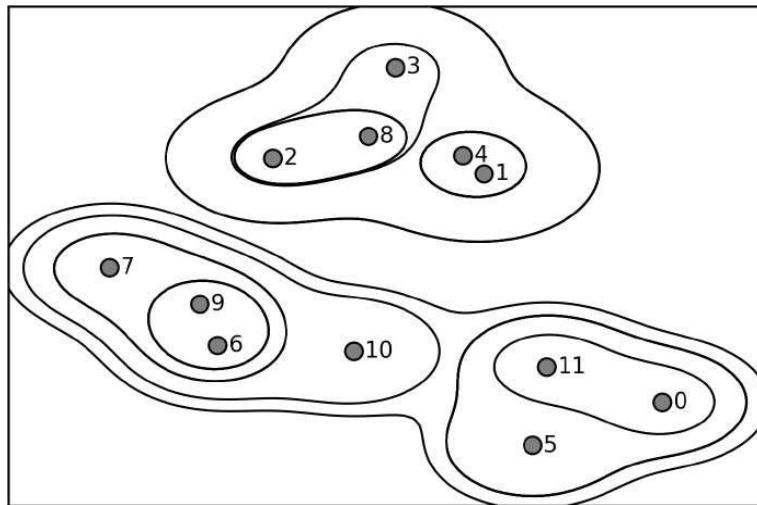
Seperti yang diharapkan, algoritma memulihkan pengelompokan dengan sempurna. Sementara implementasi scikit-learn dari pengelompokan aglomerasi mengharuskan Anda untuk menentukan jumlah klaster yang Anda inginkan untuk ditemukan oleh algoritme, metode pengelompokan aglomerasi memberikan beberapa bantuan dalam memilih nomor yang tepat, yang akan kita bahas selanjutnya.

#### Pengelompokan hierarkis dan dendrogram

Pengelompokan aglomeratif menghasilkan apa yang dikenal sebagai pengelompokan hierarkis. Pengelompokan berlangsung secara iteratif, dan setiap titik melakukan perjalanan dari cluster titik tunggal menjadi milik beberapa cluster akhir. Setiap langkah perantara menyediakan pengelompokan data (dengan jumlah cluster yang berbeda). Terkadang membantu untuk melihat semua kemungkinan pengelompokan secara bersama-sama. Contoh berikutnya (Gambar 3.35) menunjukkan overlay dari semua kemungkinan pengelompokan yang ditunjukkan pada Gambar 3.33, memberikan beberapa wawasan tentang bagaimana setiap cluster terpecah menjadi cluster yang lebih kecil:

In[63]:

```
mglearn.plots.plot_agglomerative()
```



**Gambar 3.35** Penugasan cluster hierarkis (ditampilkan sebagai garis) yang dihasilkan dengan pengelompokan aglomeratif, dengan titik data bernomor (lih. Gambar 3.36)

Sementara visualisasi ini memberikan pandangan yang sangat rinci tentang pengelompokan hierarkis, visualisasi ini bergantung pada sifat dua dimensi data dan oleh karena itu tidak dapat digunakan pada kumpulan data yang memiliki lebih dari dua fitur. Namun, ada alat lain untuk memvisualisasikan pengelompokan hierarkis, yang disebut dendrogram, yang dapat menangani kumpulan data multidimensi.

Sayangnya, scikit-learn saat ini tidak memiliki fungsi untuk menggambar dendrogram. Namun, Anda dapat membuatnya dengan mudah menggunakan SciPy. Algoritma pengelompokan SciPy memiliki antarmuka yang sedikit berbeda dengan algoritma pengelompokan scikit-learn. SciPy menyediakan fungsi yang mengambil larik data  $X$  dan menghitung larik tautan, yang mengkodekan kesamaan klaster hierarkis. Kami kemudian dapat memasukkan susunan tautan ini ke dalam fungsi dendrogram scipy untuk memplot dendrogram (Gambar 3.36):

In[64]:

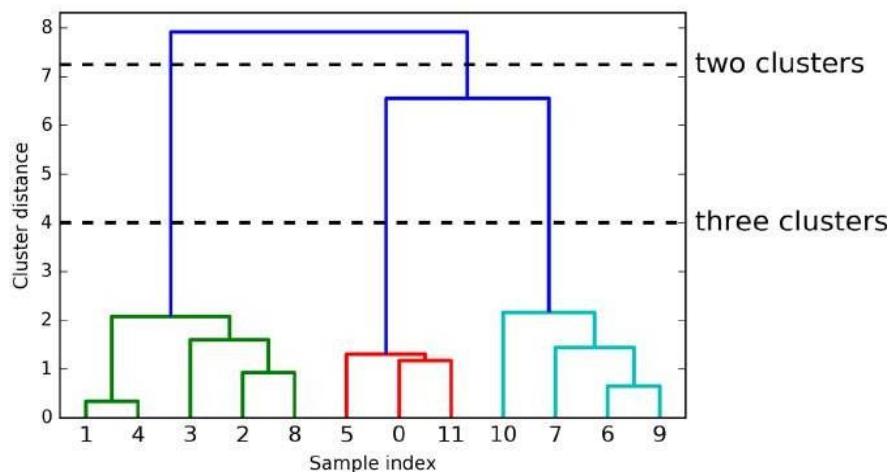
```
# Import the dendrogram function and the ward clustering function from SciPy
from scipy.cluster.hierarchy import dendrogram, ward

X, y = make_blobs(random_state=0, n_samples=12)
# Apply the ward clustering to the data array X
# The SciPy ward function returns an array that specifies the distances
# bridged when performing agglomerative clustering
linkage_array = ward(X)
```

```
# Now we plot the dendrogram for the linkage_array containing the distances
# between clusters
dendrogram(linkage_array)

# Mark the cuts in the tree that signify two or three clusters
ax = plt.gca()
bounds = ax.get_xbound()
ax.plot(bounds, [7.25, 7.25], '--', c='k')
ax.plot(bounds, [4, 4], '--', c='k')

ax.text(bounds[1], 7.25, ' two clusters', va='center', fontdict={'size': 15})
ax.text(bounds[1], 4, ' three clusters', va='center', fontdict={'size': 15})
plt.xlabel("Sample index")
plt.ylabel("Cluster distance")
```



**Gambar 3.36** Dendogram pengelompokan ditunjukkan pada Gambar 3.35 dengan garis yang menunjukkan perpecahan menjadi dua dan tiga kelompok

Dendrogram menunjukkan titik data sebagai titik di bagian bawah (dinomori dari 0 hingga 11). Kemudian, sebuah pohon diplot dengan titik-titik ini (mewakili kluster titik tunggal) sebagai daun, dan induk simpul baru ditambahkan untuk setiap dua kluster yang bergabung. Membaca dari bawah ke atas, titik data 1 dan 4 digabungkan terlebih dahulu (seperti yang Anda lihat pada Gambar 3.33). Selanjutnya, poin 6 dan 9 digabungkan menjadi sebuah cluster, dan seterusnya. Di tingkat atas, ada dua cabang, satu terdiri dari poin 11, 0, 5, 10, 7, 6, dan 9, dan yang lainnya terdiri dari poin 1, 4, 3, 2, dan 8. Ini sesuai dengan dua cluster terbesar di sisi kiri plot.

Sumbu y dalam dendrogram tidak hanya menentukan kapan dalam algoritma agglomeratif dua cluster digabungkan. Panjang setiap cabang juga menunjukkan seberapa jauh jarak cluster yang digabungkan. Cabang terpanjang dalam dendrogram ini adalah tiga garis yang ditandai dengan garis putus-putus berlabel "tiga kelompok". Bawa ini adalah cabang terpanjang menunjukkan bahwa pergi dari tiga ke dua kelompok berarti menggabungkan beberapa titik yang sangat berjauhan. Kami melihat ini lagi di bagian atas grafik, di mana menggabungkan dua cluster yang tersisa menjadi satu cluster lagi-lagi menjembatani jarak yang relatif jauh.

Sayangnya, pengelompokan aglomeratif masih gagal dalam memisahkan bentuk kompleks seperti dataset two\_moons. Tetapi hal yang sama tidak berlaku untuk algoritma berikutnya yang akan kita lihat, DBSCAN.

### 3.13 DBSCAN

Algoritma pengelompokan lain yang sangat berguna adalah DBSCAN (yang merupakan singkatan dari "pengelompokan aplikasi spasial berbasis kepadatan dengan noise"). Manfaat utama DBSCAN adalah tidak mengharuskan pengguna untuk mengatur jumlah cluster secara apriori, dapat menangkap cluster dengan bentuk yang kompleks, dan dapat mengidentifikasi titik-titik yang bukan merupakan bagian dari cluster manapun. DBSCAN agak lebih lambat daripada pengelompokan aglomeratif dan k-means, tetapi masih berskala ke kumpulan data yang relatif besar.

DBSCAN bekerja dengan mengidentifikasi titik-titik yang berada di wilayah "ramai" dari ruang fitur, di mana banyak titik data berdekatan. Daerah ini disebut sebagai daerah padat dalam ruang fitur. Ide di balik DBSCAN adalah bahwa cluster membentuk wilayah data yang padat, dipisahkan oleh wilayah yang relatif kosong.

Titik-titik yang berada dalam wilayah padat disebut sampel inti (atau titik inti), dan didefinisikan sebagai berikut. Ada dua parameter dalam DBSCAN: min\_samples dan eps. Jika setidaknya ada min\_samples banyak titik data dalam jarak eps ke titik data tertentu, titik data tersebut diklasifikasikan sebagai sampel inti. Sampel inti yang lebih dekat satu sama lain dari jarak eps dimasukkan ke dalam cluster yang sama oleh DBSCAN.

Algoritme bekerja dengan memilih titik arbitrer untuk memulai. Kemudian menemukan semua titik dengan jarak eps atau kurang dari titik itu. Jika ada kurang dari titik min\_samples dalam jarak eps dari titik awal, titik ini diberi label sebagai noise, artinya titik tersebut bukan milik cluster mana pun. Jika ada lebih dari min\_samples poin dalam jarak eps, titik tersebut diberi label sampel inti dan diberi label cluster baru. Kemudian, semua tetangga (dalam eps) dari titik tersebut dikunjungi. Jika mereka belum diberi kluster, mereka diberi label kluster baru yang baru saja dibuat. Jika mereka adalah sampel inti, tetangga mereka dikunjungi secara bergantian, dan seterusnya. Cluster tumbuh sampai tidak ada lagi sampel inti dalam jarak eps dari cluster. Kemudian titik lain yang belum dikunjungi diambil, dan prosedur yang sama diulang.

Pada akhirnya, ada tiga jenis titik: titik inti, titik yang berada dalam jarak eps dari titik inti (disebut titik batas), dan noise. Ketika algoritma DBSCAN dijalankan pada kumpulan data tertentu beberapa kali, pengelompokan titik inti selalu sama, dan titik yang sama akan selalu diberi label sebagai noise. Namun, titik batas mungkin bertetangga dengan sampel inti lebih dari satu cluster. Oleh karena itu, keanggotaan klaster dari titik-titik batas tergantung pada urutan titik-titik yang dikunjungi. Biasanya hanya ada beberapa titik batas, dan sedikit ketergantungan pada urutan titik ini tidak penting.

Mari kita terapkan DBSCAN pada dataset sintetik yang kita gunakan untuk mendemonstrasikan pengelompokan aglomeratif. Seperti pengelompokan aglomeratif, DBSCAN tidak mengizinkan prediksi pada data uji baru, jadi kami akan menggunakan metode

`fit_predict` untuk melakukan pengelompokan dan mengembalikan label klaster dalam satu langkah:

**In[65]:**

```
from sklearn.cluster import DBSCAN
X, y = make_blobs(random_state=0, n_samples=12)

dbSCAN = DBSCAN()
clusters = dbSCAN.fit_predict(X)
print("Cluster memberships:\n{}".format(clusters))
```

**Out[65]:**

```
Cluster memberships:
[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```

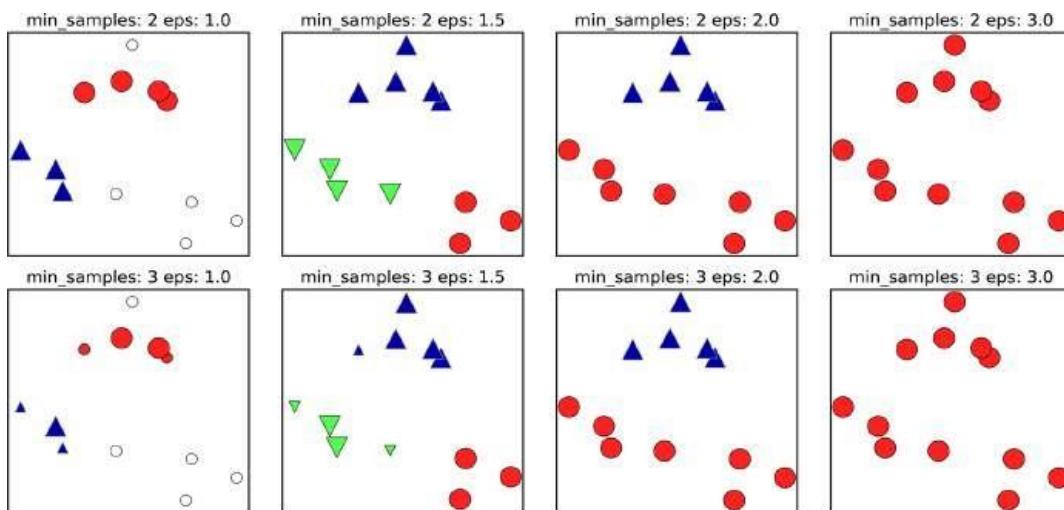
Seperti yang Anda lihat, semua titik data diberi label -1, yang merupakan singkatan dari noise. Ini adalah konsekuensi dari pengaturan parameter default untuk `eps` dan `min_samples`, yang tidak disetel untuk kumpulan data mainan kecil. Penetapan cluster untuk nilai `min_samples` dan `eps` yang berbeda ditunjukkan di bawah ini, dan divisualisasikan pada Gambar 3.37:

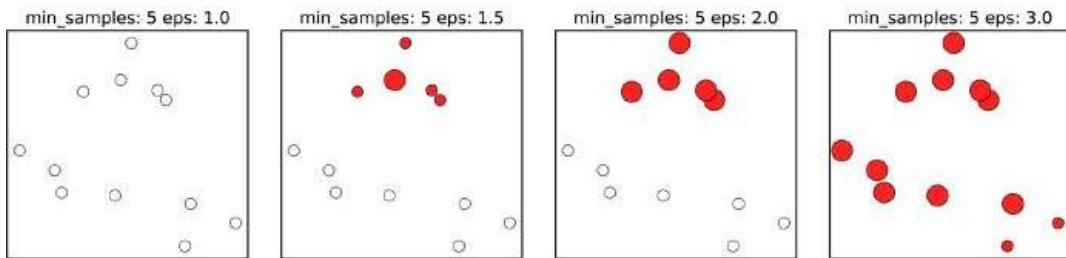
**In[66]:**

```
mglearn.plots.plot_dbSCAN()
```

**Out[66]:**

```
min_samples: 2 eps: 1.000000  cluster: [-1 0 0 -1 0 -1 1 1 1 0 1 -1 -1]
min_samples: 2 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 2 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 2 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 3 eps: 1.000000  cluster: [-1 0 0 -1 0 -1 1 1 1 0 1 -1 -1]
min_samples: 3 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 3 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 3 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 5 eps: 1.000000  cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
min_samples: 5 eps: 1.500000  cluster: [-1 0 0 0 -1 -1 -1 0 -1 -1 -1]
min_samples: 5 eps: 2.000000  cluster: [-1 0 0 0 -1 -1 -1 0 -1 -1 -1]
min_samples: 5 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
```





**Gambar 3.37** Penetapan cluster ditemukan oleh DBSCAN dengan pengaturan yang bervariasi untuk parameter `min_samples` dan `eps`

Dalam plot ini, titik-titik yang termasuk dalam cluster adalah solid, sedangkan titik-titik noise ditampilkan dalam warna putih. Sampel inti ditampilkan sebagai penanda besar, sedangkan titik batas ditampilkan sebagai penanda yang lebih kecil. Meningkatkan `eps` (dari kiri ke kanan pada gambar) berarti lebih banyak poin akan dimasukkan dalam sebuah cluster. Hal ini membuat cluster tumbuh, tetapi mungkin juga menyebabkan beberapa cluster bergabung menjadi satu. Meningkatkan `min_samples` (dari atas ke bawah pada gambar) berarti lebih sedikit poin yang akan menjadi poin inti, dan lebih banyak poin akan diberi label sebagai noise.

Parameter `eps` agak lebih penting, karena menentukan apa artinya poin menjadi "dekat." Menyetel `eps` menjadi sangat kecil akan berarti bahwa tidak ada titik yang merupakan sampel inti, dan dapat menyebabkan semua titik diberi label sebagai noise. Setting `eps` menjadi sangat besar akan mengakibatkan semua titik membentuk satu cluster.

Pengaturan `min_samples` sebagian besar menentukan apakah titik di wilayah yang kurang padat akan diberi label sebagai outlier atau sebagai klusternya sendiri. Jika Anda mengurangi `min_samples`, apa pun yang akan menjadi cluster dengan kurang dari `min_samples` banyak sampel sekarang akan diberi label sebagai noise. `min_samples` karena itu menentukan ukuran cluster minimum. Anda dapat melihat ini dengan sangat jelas pada Gambar 3.37, ketika beralih dari `min_samples=3` ke `min_samples=5` dengan `eps=1.5`. Dengan `min_samples=3`, ada tiga cluster: satu dari empat poin, satu dari lima poin, dan satu dari tiga poin. Menggunakan `min_samples=5`, dua cluster yang lebih kecil (dengan tiga dan empat titik) sekarang diberi label sebagai noise, dan hanya cluster dengan lima sampel yang tersisa.

Sementara DBSCAN tidak memerlukan pengaturan jumlah cluster secara eksplisit, pengaturan `eps` secara implisit mengontrol berapa banyak cluster yang akan ditemukan. Menemukan pengaturan yang baik untuk `eps` terkadang lebih mudah setelah menskalakan data menggunakan StandardScaler atau MinMaxScaler, karena menggunakan teknik penskalaan ini akan memastikan bahwa semua fitur memiliki rentang yang sama.

Gambar 3.38 menunjukkan hasil menjalankan DBSCAN pada dataset two\_moons. Algoritme sebenarnya menemukan dua setengah lingkaran dan memisahkannya menggunakan pengaturan default:

In[67]:

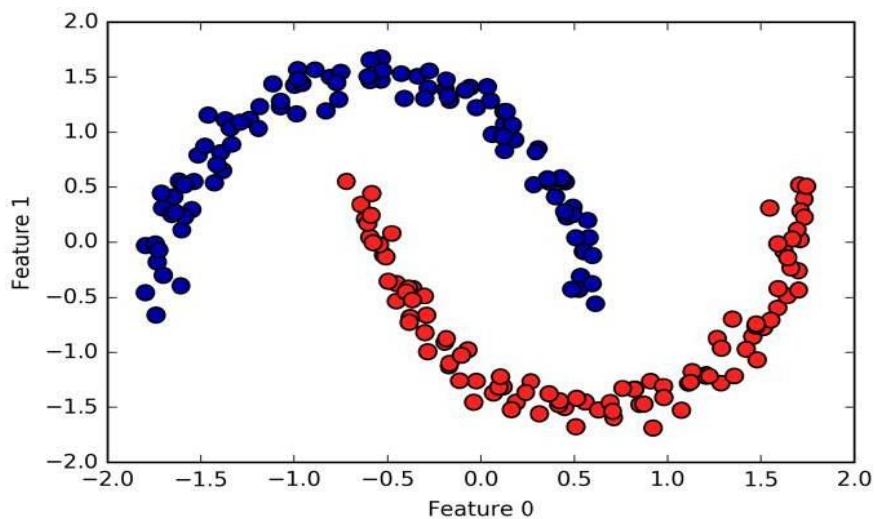
```
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# rescale the data to zero mean and unit variance
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

dbSCAN = DBSCAN()
clusters = dbSCAN.fit_predict(X_scaled)
# plot the cluster assignments
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm2, s=60)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Karena algoritma menghasilkan jumlah cluster (dua) yang diinginkan, pengaturan parameter tampaknya bekerja dengan baik. Jika kita menurunkan eps ke 0.2 (dari default 0.5), kita akan mendapatkan delapan cluster, yang jelas terlalu banyak. Meningkatkan eps ke 0,7 menghasilkan satu cluster.

Saat menggunakan DBSCAN, Anda harus berhati-hati dalam menangani tugas cluster yang dikembalikan. Penggunaan -1 untuk menunjukkan noise dapat mengakibatkan efek yang tidak terduga saat menggunakan label cluster untuk mengindeks array lain.



**Gambar 3.38** Penetapan cluster ditemukan oleh DBSCAN menggunakan nilai default  $\text{eps}=0.5$

### 3.14 MEMBANDINGKAN DAN MENGEVALUASI ALGORITMA CLUSTERING

Salah satu tantangan dalam menerapkan algoritma pengelompokan adalah sangat sulit untuk menilai seberapa baik suatu algoritma bekerja, dan untuk membandingkan hasil antara algoritma yang berbeda. Setelah berbicara tentang algoritma di balik k-means, agglomerative clustering, dan DBSCAN, sekarang kita akan membandingkannya pada beberapa dataset dunia nyata.

## Mengevaluasi pengelompokan dengan kebenaran dasar

Ada metrik yang dapat digunakan untuk menilai hasil dari algoritma pengelompokan relatif terhadap pengelompokan kebenaran dasar, yang paling penting adalah indeks rand yang disesuaikan (ARI) dan informasi timbal balik yang dinormalisasi (NMI), yang keduanya memberikan ukuran kuantitatif antara 0 dan 1.

Di sini, kami membandingkan algoritma k-means, agglomerative clustering, dan DBSCAN menggunakan ARI. Kami juga menyertakan tampilannya ketika kami secara acak menetapkan titik ke dua cluster untuk perbandingan (lihat Gambar 3.39):

In[68]:

```
from sklearn.metrics.cluster import adjusted_rand_score
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# rescale the data to zero mean and unit variance
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

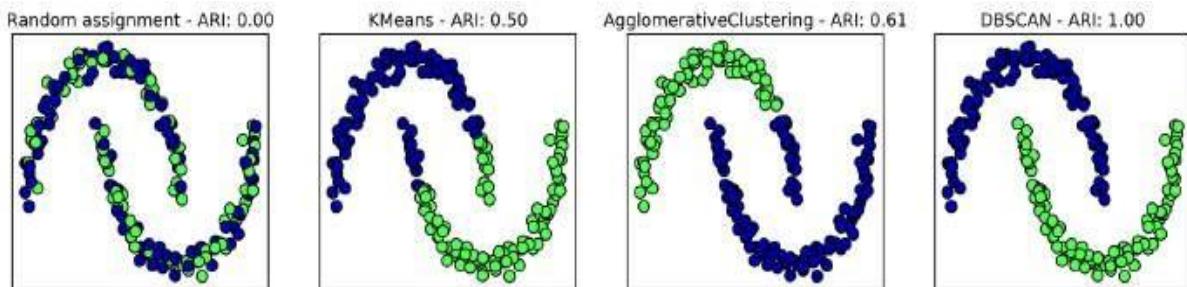
fig, axes = plt.subplots(1, 4, figsize=(15, 3),
                       subplot_kw={'xticks': (), 'yticks': ()})

# make a list of algorithms to use
algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2),
              DBSCAN()]

# create a random cluster assignment for reference
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

# plot random assignment
axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters,
                 cmap=mlearn.cm3, s=60)
axes[0].set_title("Random assignment - ARI: {:.2f}".format(
    adjusted_rand_score(y, random_clusters)))

for ax, algorithm in zip(axes[1:], algorithms):
    # plot the cluster assignments and cluster centers
    clusters = algorithm.fit_predict(X_scaled)
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters,
               cmap=mlearn.cm3, s=60)
    ax.set_title("{} - ARI: {:.2f}".format(algorithm.__class__.__name__,
                                            adjusted_rand_score(y, clusters)))
```



**Gambar 3.39** Membandingkan penugasan acak, k-means, agglomerative clustering, dan DBSCAN pada dataset two\_moons menggunakan skor ARI terawasi

Indeks rand yang disesuaikan memberikan hasil yang intuitif, dengan penugasan cluster acak memiliki skor 0 dan DBSCAN (yang memulihkan pengelompokan yang diinginkan dengan sempurna) memiliki skor 1.

Kesalahan umum saat mengevaluasi pengelompokan dengan cara ini adalah menggunakan akurasi\_score alih-alih adjusted\_rand\_score, normalized\_mutual\_info\_score, atau metrik pengelompokan lainnya. Masalah dalam menggunakan akurasi adalah membutuhkan label cluster yang ditetapkan untuk benar-benar cocok dengan kebenaran dasar. Namun, label cluster itu sendiri tidak ada artinya—satu-satunya hal yang penting adalah titik mana yang berada di cluster yang sama:

**In[69]:**

```
from sklearn.metrics import accuracy_score

# these two labelings of points correspond to the same clustering
clusters1 = [0, 0, 1, 1, 0]
clusters2 = [1, 1, 0, 0, 1]
# accuracy is zero, as none of the labels are the same
print("Accuracy: {:.2f}".format(accuracy_score(clusters1, clusters2)))
# adjusted rand score is 1, as the clustering is exactly the same
print("ARI: {:.2f}".format(adjusted_rand_score(clusters1, clusters2)))
```

**Out[69]:**

```
Accuracy: 0.00
ARI: 1.00
```

### Mengevaluasi pengelompokan tanpa kebenaran dasar

Meskipun kami baru saja menunjukkan satu cara untuk mengevaluasi algoritma pengelompokan, dalam praktiknya, ada masalah besar dengan menggunakan ukuran seperti ARI. Saat menerapkan algoritma pengelompokan, biasanya tidak ada kebenaran dasar untuk membandingkan hasilnya. Jika kita mengetahui pengelompokan data yang tepat, kita dapat menggunakan informasi ini untuk membangun model yang diawasi seperti pengklasifikasi. Oleh karena itu, menggunakan metrik seperti ARI dan NMI biasanya hanya membantu dalam mengembangkan algoritme, bukan dalam menilai keberhasilan dalam suatu aplikasi.

Ada metrik penilaian untuk pengelompokan yang tidak memerlukan kebenaran dasar, seperti koefisien siluet. Namun, ini sering tidak berfungsi dengan baik dalam praktiknya. Skor siluet menghitung kekompakan sebuah cluster, di mana semakin tinggi semakin baik, dengan skor sempurna 1. Sementara cluster kompak baik, kekompakan tidak memungkinkan untuk bentuk yang rumit.

Berikut adalah contoh perbandingan hasil k-means, agglomerative clustering, dan DBSCAN pada dataset dua bulan menggunakan nilai siluet (Gambar 3.40):

**In[70]:**

```
from sklearn.metrics.cluster import silhouette_score

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
# rescale the data to zero mean and unit variance
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```

fig, axes = plt.subplots(1, 4, figsize=(15, 3),
                      subplot_kw={'xticks': (), 'yticks': ()})

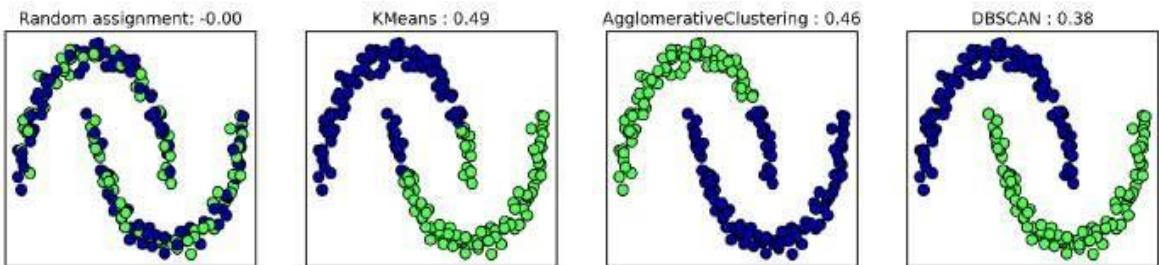
# create a random cluster assignment for reference
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

# plot random assignment
axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters,
                 cmap=mlearn.cm3, s=60)
axes[0].set_title("Random assignment: {:.2f}".format(
    silhouette_score(X_scaled, random_clusters)))

algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2),
              DBSCAN()]

for ax, algorithm in zip(axes[1:], algorithms):
    clusters = algorithm.fit_predict(X_scaled)
    # plot the cluster assignments and cluster centers
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mlearn.cm3,
               s=60)
    ax.set_title("{} : {:.2f}".format(algorithm.__class__.__name__,
                                      silhouette_score(X_scaled, clusters)))

```



**Gambar 3.40** Membandingkan penugasan acak, k-means, agglomerative clustering, dan DBSCAN pada dataset two\_moons menggunakan skor siluet unsupervised—hasil DBSCAN yang lebih intuitif memiliki skor siluet yang lebih rendah daripada penugasan yang ditemukan oleh k-means

Seperti yang Anda lihat, k-means mendapatkan skor siluet tertinggi, meskipun kami mungkin lebih menyukai hasil yang dihasilkan oleh DBSCAN. Strategi yang sedikit lebih baik untuk mengevaluasi cluster adalah menggunakan metrik clustering berbasis ketahanan. Ini menjalankan algoritme setelah menambahkan beberapa noise ke data, atau menggunakan pengaturan parameter yang berbeda, dan membandingkan hasilnya. Ideanya adalah bahwa jika banyak parameter algoritme dan banyak gangguan data mengembalikan hasil yang sama, kemungkinan besar dapat dipercaya. Sayangnya, strategi ini tidak diterapkan dalam scikit-learn pada saat penulisan.

Bahkan jika kita mendapatkan pengelompokan yang sangat kuat, atau skor siluet yang sangat tinggi, kita masih tidak tahu apakah ada makna semantik dalam pengelompokan, atau apakah pengelompokan tersebut mencerminkan aspek data yang kita minati. Ayo pergi kembali ke contoh gambar wajah. Kami berharap menemukan kelompok wajah yang mirip—

katakanlah, pria dan wanita, atau orang tua dan orang muda, atau orang dengan janggut dan tanpa janggut. Katakanlah kita mengelompokkan data menjadi dua kelompok, dan semua algoritma setuju tentang titik mana yang harus dikelompokkan bersama. Kami masih tidak tahu apakah cluster yang ditemukan sesuai dengan konsep yang kami minati. Bisa jadi mereka menemukan tampilan samping versus tampilan depan, atau gambar yang diambil pada malam hari versus gambar yang diambil pada siang hari, atau gambar - gambar yang diambil dengan iPhone versus gambar yang diambil dengan ponsel Android. Satu-satunya cara untuk mengetahui apakah pengelompokan sesuai dengan apa pun yang kami minati adalah dengan menganalisis pengelompokan secara manual.

### Membandingkan algoritma pada dataset wajah

Mari kita terapkan algoritma k-means, DBSCAN, dan agglomerative clustering ke dataset Labeled Faces in the Wild, dan lihat apakah ada di antara mereka yang menemukan struktur yang menarik. Kami akan menggunakan representasi eigenface dari data, seperti yang dihasilkan oleh PCA (whiten=True), dengan 100 komponen:

**In[71]:**

```
# extract eigenfaces from lfw data and transform data
from sklearn.decomposition import PCA
pca = PCA(n_components=100, whiten=True, random_state=0)
pca.fit_transform(X_people)
X_pca = pca.transform(X_people)
```

Kita telah melihat sebelumnya bahwa ini adalah representasi yang lebih semantik dari gambar wajah daripada piksel mentah. Ini juga akan membuat komputasi lebih cepat. Latihan yang baik adalah Anda menjalankan eksperimen berikut pada data asli, tanpa PCA, dan melihat apakah Anda menemukan cluster yang serupa.

Menganalisis dataset wajah dengan DBSCAN. Kita akan mulai dengan menerapkan DBSCAN, yang baru saja kita bahas:

**In[72]:**

```
# apply DBSCAN with default parameters
dbSCAN = DBSCAN()
labels = dbSCAN.fit_predict(X_pca)
print("Unique labels: {}".format(np.unique(labels)))
```

**Out[72]:**

```
Unique labels: [-1]
```

Kami melihat bahwa semua label yang dikembalikan adalah -1, jadi semua data diberi label sebagai "noise" oleh DBSCAN. Ada dua hal yang dapat kita ubah untuk membantu ini: kita dapat membuat eps lebih tinggi, memperluas lingkungan setiap titik, dan menetapkan min\_samples lebih rendah, untuk mempertimbangkan kelompok titik yang lebih kecil sebagai cluster. Mari kita coba mengubah min\_samples terlebih dahulu:

**In[73]:**

```
dbSCAN = DBSCAN(min_samples=3)
labels = dbSCAN.fit_predict(X_pca)
print("Unique labels: {}".format(np.unique(labels)))
```

**Out[73]:**

```
Unique labels: [-1]
```

Bahkan ketika mempertimbangkan kelompok tiga poin, semuanya diberi label sebagai kebisingan. Jadi, kita perlu meningkatkan `eps`:

**In[74]:**

```
dbSCAN = DBSCAN(min_samples=3, eps=15)
labels = dbSCAN.fit_predict(X_pca)
print("Unique labels: {}".format(np.unique(labels)))
```

**Out[74]:**

```
Unique labels: [-1  0]
```

Menggunakan `eps` 15 yang jauh lebih besar, kami hanya mendapatkan satu cluster dan titik noise. Kita dapat menggunakan hasil ini untuk mengetahui seperti apa “noise” tersebut dibandingkan dengan data lainnya. Untuk lebih memahami apa yang terjadi, mari kita lihat berapa banyak titik yang noise, dan berapa banyak titik yang ada di dalam cluster:

**In[75]:**

```
# Count number of points in all clusters and noise.
# bincount doesn't allow negative numbers, so we need to add 1.
# The first number in the result corresponds to noise points.
print("Number of points per cluster: {}".format(np.bincount(labels + 1)))
```

**Out[75]:**

```
Number of points per cluster: [ 27 2036]
```

Ada sangat sedikit titik noise—hanya 27—sehingga kita dapat melihat semuanya (lihat Gambar 3.41):

**In[76]:**

```
noise = X_people[labels == -1]

fig, axes = plt.subplots(3, 9, subplot_kw={'xticks': (), 'yticks': ()},
                       figsize=(12, 4))
for image, ax in zip(noise, axes.ravel()):
    ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
```



**Gambar 3.41** Sampel dari dataset wajah yang diberi label sebagai noise oleh DBSCAN

Membandingkan gambar-gambar ini dengan sampel acak gambar wajah dari Gambar 3.7, kita dapat menebak mengapa mereka diberi label sebagai noise: gambar kelima di baris pertama menunjukkan orang yang minum dari gelas, ada gambar orang yang memakai topi., dan pada gambar terakhir ada tangan di depan wajah orang tersebut. Gambar lainnya berisi sudut aneh atau potongan yang terlalu dekat atau terlalu lebar.

Analisis semacam ini—mencoba menemukan “yang aneh”—disebut deteksi outlier. Jika ini adalah aplikasi nyata, kami mungkin mencoba melakukan pekerjaan cropping gambar yang lebih baik, untuk mendapatkan data yang lebih homogen. Tidak banyak yang dapat kami lakukan tentang orang-orang di foto yang terkadang mengenakan topi, minum, atau memegang sesuatu di depan wajah mereka, tetapi ada baiknya mengetahui bahwa ini adalah masalah dalam data yang perlu ditangani oleh algoritme apa pun yang mungkin kami terapkan.

Jika kita ingin menemukan cluster yang lebih menarik daripada hanya satu cluster besar, kita perlu mengatur eps lebih kecil, antara 15 dan 0,5 (default). Mari kita lihat perbedaan nilai eps yang dihasilkan:

**In[77]:**

```
for eps in [1, 3, 5, 7, 9, 11, 13]:
    print("\neps={}".format(eps))
    dbscan = DBSCAN(eps=eps, min_samples=3)
    labels = dbscan.fit_predict(X_pca)
    print("Clusters present: {}".format(np.unique(labels)))
    print("Cluster sizes: {}".format(np.bincount(labels + 1)))
```

**Out[78]:**

```
eps=1
Clusters present: [-1]
Cluster sizes: [2063]

eps=3
Clusters present: [-1]
Cluster sizes: [2063]
```

```

eps=5
Clusters present: [-1]
Cluster sizes: [2063]

eps=7
Clusters present: [-1  0  1  2  3  4  5  6  7  8  9 10 11 12]
Cluster sizes: [2006  4  6  6  6  9  3  3  4  3  3  3  3  3  4]

eps=9
Clusters present: [-1  0  1  2]
Cluster sizes: [1269  788    3    3]

eps=11
Clusters present: [-1  0]
Cluster sizes: [ 430 1633]

eps=13
Clusters present: [-1  0]
Cluster sizes: [ 112 1951]

```

Untuk pengaturan eps rendah, semua titik diberi label sebagai noise. Untuk eps=7, kami mendapatkan banyak titik noise dan banyak cluster yang lebih kecil. Untuk eps=9 kita masih mendapatkan banyak noise point, tetapi kita mendapatkan satu cluster besar dan beberapa cluster yang lebih kecil. Mulai dari eps=11, kita hanya mendapatkan satu cluster besar dan noise.

Yang menarik untuk dicatat adalah bahwa tidak pernah ada lebih dari satu cluster besar. Paling banyak, ada satu cluster besar yang berisi sebagian besar titik, dan beberapa cluster yang lebih kecil. Hal ini menunjukkan bahwa tidak ada dua atau tiga jenis citra wajah yang berbeda dalam data yang sangat berbeda, melainkan bahwa semua citra kurang lebih sama dengan (atau berbeda dari) yang lain.

Hasil untuk eps=7 terlihat paling menarik, dengan banyak cluster kecil. Kita dapat menyelidiki pengelompokan ini secara lebih rinci dengan memvisualisasikan semua titik di masing-masing dari 13 kelompok kecil (Gambar 3.42):

In[78]:

```

dbSCAN = DBSCAN(min_samples=3, eps=7)
labels = dbSCAN.fit_predict(X_pca)

for cluster in range(max(labels) + 1):
    mask = labels == cluster
    n_images = np.sum(mask)
    fig, axes = plt.subplots(1, n_images, figsize=(n_images * 1.5, 4),
                           subplot_kw={'xticks': (), 'yticks': ()})
    for image, label, ax in zip(X_people[mask], y_people[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1])

```



**Gambar 3.42** Cluster ditemukan oleh DBSCAN dengan  $\text{eps}=7$

Beberapa cluster sesuai dengan orang-orang dengan wajah yang sangat berbeda (dalam kumpulan data ini), seperti Sharon atau Koizumi. Dalam setiap cluster, orientasi wajah juga cukup tetap, begitu juga dengan ekspresi wajah. Beberapa cluster berisi wajah beberapa orang, tetapi mereka memiliki orientasi dan ekspresi yang sama.

Ini menyimpulkan analisis kami tentang algoritma DBSCAN yang diterapkan pada dataset wajah. Seperti yang Anda lihat, kami melakukan analisis manual di sini, berbeda dari pendekatan pencarian otomatis yang jauh lebih banyak yang dapat kami gunakan untuk pembelajaran terawasi berdasarkan skor atau akurasi R2. Mari kita beralih ke penerapan k-means dan agglomerative clustering.

### Menganalisis dataset wajah dengan k-means.

Kami melihat bahwa tidak mungkin membuat lebih dari satu cluster besar menggunakan DBSCAN. Pengelompokan aglomeratif dan k-means jauh lebih mungkin untuk membuat cluster dengan ukuran yang sama, tetapi kita perlu menetapkan jumlah target cluster. Kami dapat mengatur jumlah cluster ke jumlah orang yang diketahui dalam kumpulan data, meskipun sangat tidak mungkin bahwa algoritma pengelompokan yang tidak diawasi akan memulihkannya. Sebagai gantinya, kita bisa mulai dengan jumlah cluster yang sedikit, seperti 10, yang memungkinkan kita untuk menganalisis setiap cluster:

**In[79]:**

```
# extract clusters with k-means
km = KMeans(n_clusters=10, random_state=0)
labels_km = km.fit_predict(X_pca)
print("Cluster sizes k-means: {}".format(np.bincount(labels_km)))
```

**Out[79]:**

```
Cluster sizes k-means: [269 128 170 186 386 222 237 64 253 148]
```

Seperti yang Anda lihat, k-means clustering mempartisi data ke dalam cluster berukuran relatif sama dari 64 hingga 386. Ini sangat berbeda dari hasil DBSCAN.

Kita dapat menganalisis lebih lanjut hasil k-means dengan memvisualisasikan pusat cluster (Gambar 3.43). Saat kami mengelompokkan representasi yang dihasilkan oleh PCA, kami perlu memutar pusat cluster kembali ke ruang asli untuk memvisualisasikannya, menggunakan `pca.inverse_transform`:

**In[80]:**

```
fig, axes = plt.subplots(2, 5, subplot_kw={'xticks': (), 'yticks': ()},
                       figsize=(12, 4))
for center, ax in zip(km.cluster_centers_, axes.ravel()):
    ax.imshow(pca.inverse_transform(center).reshape(image_shape),
              vmin=0, vmax=1)
```



**Gambar 3.43** Pusat cluster ditemukan oleh k-means saat mengatur jumlah cluster menjadi

Pusat cluster yang ditemukan oleh k-means adalah versi wajah yang sangat halus. Ini tidak terlalu mengejutkan, mengingat setiap pusat rata-rata memiliki 64 hingga 386 gambar wajah. Bekerja dengan representasi PCA yang dikurangi menambah kehalusan gambar (dibandingkan dengan wajah yang direkonstruksi menggunakan 100 dimensi PCA pada Gambar 3.11). Pengelompokan tampaknya mengambil orientasi wajah yang berbeda, ekspresi yang berbeda (pusat cluster ketiga tampaknya menunjukkan wajah tersenyum), dan kehadiran kerah kemeja (lihat pusat cluster kedua dari terakhir).

Untuk tampilan yang lebih rinci, pada Gambar 3.44 kami menunjukkan untuk setiap pusat cluster lima gambar paling khas di cluster (gambar ditugaskan ke cluster yang paling dekat dengan pusat cluster) dan lima gambar paling atipikal di cluster (gambar yang ditugaskan ke cluster yang terjauh dari pusat cluster):

**In[81]:**

```
mlearn.plots.plot_kmeans_faces(km, pca, X_pca, X_people,
                                y_people, people.target_names)
```





**Gambar 3.44** Contoh gambar untuk setiap cluster yang ditemukan dengan k-means—pusat cluster berada di sebelah kiri, diikuti oleh lima titik terdekat ke setiap pusat dan lima titik yang ditetapkan ke cluster tetapi terjauh dari pusat

Gambar 3.44 menegaskan intuisi kita tentang wajah tersenyum untuk cluster ketiga, dan juga pentingnya orientasi untuk cluster lainnya. Namun, titik-titik “atipikal” tidak terlalu mirip dengan pusat-pusat cluster, dan penugasannya tampaknya agak sewenang-wenang. Ini dapat dikaitkan dengan fakta bahwa k-means mempartisi semua titik data dan tidak memiliki konsep titik “noise”, seperti yang dilakukan DBSCAN. Menggunakan jumlah cluster yang lebih besar, algoritme dapat menemukan perbedaan yang lebih baik. Namun, menambahkan lebih banyak cluster membuat pemeriksaan manual menjadi lebih sulit.

Menganalisis dataset wajah dengan pengelompokan agglomeratif. Sekarang, mari kita lihat hasil dari agglomerative clustering:

**In[82]:**

```
# extract clusters with ward agglomerative clustering
agglomerative = AgglomerativeClustering(n_clusters=10)
labels_agg = agglomerative.fit_predict(X_pca)
print("Cluster sizes agglomerative clustering: {}".format(
    np.bincount(labels_agg)))
```

**Out[82]:**

```
Cluster sizes agglomerative clustering: [255 623  86 102 122 199 265  26 230 155]
```

Pengelompokan agglomeratif juga menghasilkan klaster yang berukuran relatif sama, dengan ukuran klaster antara 26 dan 623. Ini lebih tidak merata daripada yang dihasilkan oleh k-means, tetapi jauh lebih merata daripada yang dihasilkan oleh DBSCAN.

Kita dapat menghitung ARI untuk mengukur apakah dua partisi data yang diberikan oleh pengelompokan agglomeratif dan k-means serupa:

**In[83]:**

```
print("ARI: {:.2f}".format(adjusted_rand_score(labels_agg, labels_km)))
```

**Out[83]:**

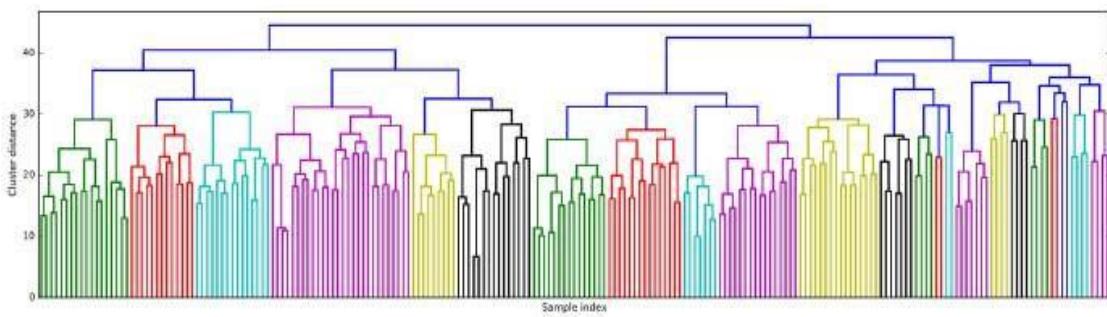
```
ARI: 0.13
```

ARI hanya 0,13 berarti bahwa dua pengelompokan labels\_agg dan labels\_km memiliki sedikit kesamaan. Ini tidak terlalu mengejutkan, mengingat fakta bahwa titik yang lebih jauh dari pusat cluster tampaknya memiliki sedikit kesamaan untuk k-means.

Selanjutnya, kita mungkin ingin memplot dendrogram (Gambar 3.45). Kami akan membatasi kedalaman pohon di plot, karena bercabang ke 2.063 titik data individual akan menghasilkan plot padat yang tidak terbaca:

**In[84]:**

```
linkage_array = ward(X_pca)
# now we plot the dendrogram for the linkage_array
# containing the distances between clusters
plt.figure(figsize=(20, 5))
dendrogram(linkage_array, p=7, truncate_mode='level', no_labels=True)
plt.xlabel("Sample index")
plt.ylabel("Cluster distance")
```



**Gambar 3.45** Dendrogram pengelompokan aglomeratif pada kumpulan data wajah

Membuat 10 cluster, kami memotong pohon di bagian paling atas, di mana ada 10 garis vertikal. Dalam dendrogram untuk data mainan yang ditunjukkan pada Gambar 3.36, Anda dapat melihat dari panjang cabang bahwa dua atau tiga kelompok dapat menangkap data dengan tepat. Untuk data wajah, sepertinya tidak ada titik potong yang sangat alami. Ada beberapa cabang yang mewakili grup yang lebih berbeda, tetapi tampaknya tidak ada jumlah cluster tertentu yang cocok. Ini tidak mengherankan, mengingat hasil DBSCAN, yang mencoba mengelompokkan semua titik menjadi satu.

Mari kita visualisasikan 10 cluster, seperti yang kita lakukan untuk k-means sebelumnya (Gambar 3.46). Perhatikan bahwa tidak ada gagasan tentang pusat klaster dalam pengelompokan aglomerat (walaupun kita dapat menghitung rata-ratanya), dan kita hanya menunjukkan beberapa titik pertama di setiap klaster. Kami menunjukkan jumlah titik di setiap cluster di sebelah kiri gambar pertama:

**In[85]:**

```
n_clusters = 10
for cluster in range(n_clusters):
    mask = labels_agg == cluster
    fig, axes = plt.subplots(1, 10, subplot_kw={'xticks': (), 'yticks': ()},
                           figsize=(15, 8))
    axes[0].set_ylabel(np.sum(mask))
    for image, label, asdf, ax in zip(X_people[mask], y_people[mask],
                                      labels_agg[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1],
                    fontdict={'fontsize': 9})
```



**Gambar 3.46** Gambar acak dari cluster yang dihasilkan oleh In[82]—setiap baris sesuai dengan satu cluster; nomor di sebelah kiri mencantumkan jumlah gambar di setiap cluster

Sementara beberapa cluster tampaknya memiliki tema semantik, banyak dari mereka terlalu besar untuk benar-benar homogen. Untuk mendapatkan cluster yang lebih homogen, kita dapat menjalankan algoritme lagi, kali ini dengan 40 cluster, dan memilih beberapa cluster yang sangat menarik (Gambar 3.47):

In[86]:

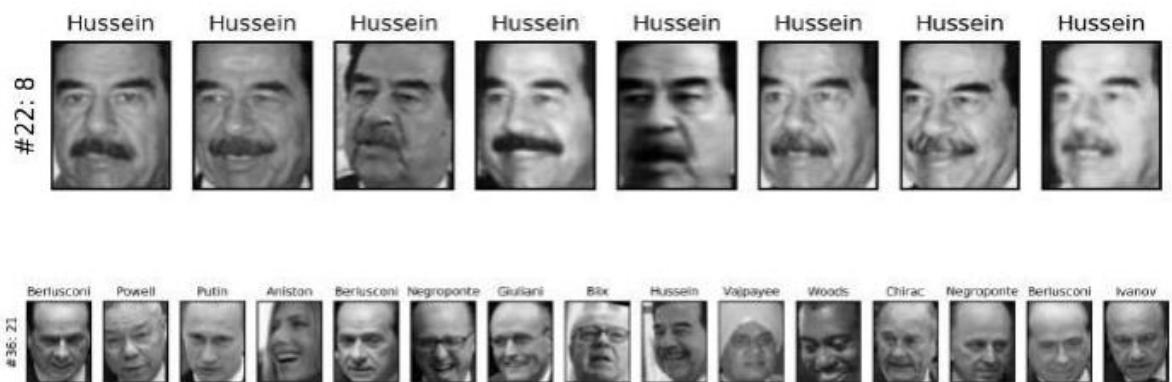
```
# extract clusters with ward agglomerative clustering
agglomerative = AgglomerativeClustering(n_clusters=40)
labels_agg = agglomerative.fit_predict(X_pca)
print("cluster sizes agglomerative clustering: {}".format(np.bincount(labels_agg)))

n_clusters = 40
for cluster in [10, 13, 19, 22, 36]: # hand-picked "interesting" clusters
    mask = labels_agg == cluster
    fig, axes = plt.subplots(1, 15, subplot_kw={'xticks': (), 'yticks': ()},
                           figsize=(15, 8))
    cluster_size = np.sum(mask)
    axes[0].set_ylabel("#{}: {}".format(cluster, cluster_size))
    for image, label, ax in zip(X_people[mask], y_people[mask],
                                labels_agg[mask], axes):
        ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
        ax.set_title(people.target_names[label].split()[-1],
                     fontdict={'fontsize': 9})
    for i in range(cluster_size, 15):
        axes[i].set_visible(False)
```

Out[86]:

```
cluster sizes agglomerative clustering:
[ 58  80  79  40 222  50  55  78 172  28  26  34  14  11  60  66 152  27
 47  31  54   5   8  56   3   5   8  18  22  82  37  89  28  24  41  40
 21  10 113  69]
```





**Gambar 3.47** Gambar dari klaster terpilih ditemukan oleh pengelompokan aglomerat saat menyetel jumlah klaster menjadi 40—teks di sebelah kiri menunjukkan indeks klaster dan jumlah total titik dalam klaster

Di sini, pengelompokan tampaknya telah mengambil "berkulit gelap dan tersenyum," "kemeja kemeja," "wanita tersenyum," "Hussein," dan "dahi tinggi." Kami juga dapat menemukan kluster yang sangat mirip ini menggunakan dendrogram, jika kami melakukan analisis lebih rinci.

### 3.15 RINGKASAN

Bagian ini telah menunjukkan bahwa menerapkan dan mengevaluasi pengelompokan adalah prosedur yang sangat kualitatif, dan sering kali paling membantu dalam fase eksplorasi analisis data. Kami melihat tiga algoritma pengelompokan: k-means, DBSCAN, dan pengelompokan aglomerasi. Ketiganya memiliki cara untuk mengontrol granularity clustering. k-means dan agglomerative clustering memungkinkan Anda untuk menentukan jumlah cluster yang diinginkan, sementara DBSCAN memungkinkan Anda menentukan proximity menggunakan parameter  $\text{eps}$ , yang secara tidak langsung mempengaruhi ukuran cluster. Ketiga metode tersebut dapat digunakan pada kumpulan data dunia nyata yang besar, relatif mudah dipahami, dan memungkinkan pengelompokan ke dalam banyak klaster.

Masing-masing algoritma memiliki kekuatan yang agak berbeda. k-means memungkinkan untuk karakterisasi cluster menggunakan cluster berarti. Ini juga dapat dilihat sebagai metode dekomposisi, di mana setiap titik data diwakili oleh pusat clusternya. DBSCAN memungkinkan pendekripsi "titik kebingungan" yang tidak ditetapkan pada klaster mana pun, dan dapat membantu menentukan jumlah klaster secara otomatis. Berbeda dengan dua metode lainnya, ini memungkinkan bentuk cluster yang kompleks, seperti yang kita lihat dalam contoh `two_moons`. DBSCAN terkadang menghasilkan cluster dengan ukuran yang sangat berbeda, yang dapat menjadi kekuatan atau kelemahan. Pengelompokan aglomeratif dapat menyediakan seluruh hierarki kemungkinan partisi data, yang dapat dengan mudah diperiksa melalui dendrogram.

#### Ringkasan dan Pandangan

Bab ini memperkenalkan berbagai algoritma pembelajaran tanpa pengawasan yang dapat diterapkan untuk analisis data eksplorasi dan pemrosesan awal. Memiliki representasi

data yang tepat seringkali penting untuk keberhasilan pembelajaran yang diawasi atau tidak, dan metode pra-pemrosesan dan dekomposisi memainkan peran penting dalam persiapan data.

Dekomposisi, pembelajaran manifold, dan pengelompokan adalah alat penting untuk meningkatkan pemahaman Anda tentang data Anda, dan dapat menjadi satu-satunya cara untuk memahami data Anda tanpa adanya informasi pengawasan. Bahkan dalam pengaturan yang diawasi, alat eksplorasi penting untuk pemahaman yang lebih baik tentang properti data. Seringkali sulit untuk mengukur kegunaan algoritme yang tidak diawasi, meskipun ini seharusnya tidak menghalangi Anda untuk menggunakan untuk mengumpulkan wawasan dari data Anda. Dengan metode ini di bawah ikat pinggang Anda, Anda sekarang dilengkapi dengan semua algoritme pembelajaran penting yang digunakan oleh praktisi pembelajaran mesin setiap hari.

Kami mendorong Anda untuk mencoba metode pengelompokan dan dekomposisi baik pada data mainan dua dimensi maupun pada kumpulan data dunia nyata yang termasuk dalam scikit-learn, seperti kumpulan data digit, iris, dan kanker.

### **Ringkasan Antarmuka Penaksir**

Mari kita tinjau secara singkat API yang kita perkenalkan di Bab 2 dan 3. Semua algoritme dalam scikit-learn, baik algoritme prapemrosesan, pembelajaran terawasi, atau pembelajaran tanpa pengawasan, diimplementasikan sebagai kelas. Kelas-kelas ini disebut estimator dalam scikit-belajar. Untuk menerapkan algoritma, pertama-tama Anda harus membuat instance objek dari kelas tertentu:

**In[87]:**

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

Kelas estimator berisi algoritma, dan juga menyimpan model yang dipelajari dari data menggunakan algoritma.

Anda harus menetapkan parameter model apa pun saat membuat objek model. Parameter ini termasuk regularisasi, kontrol kompleksitas, jumlah cluster untuk ditemukan, dll. Semua estimator memiliki metode fit, yang digunakan untuk membangun model. Metode fit selalu membutuhkan sebagai argumen pertama data X, direpresentasikan sebagai array NumPy atau matriks sparse SciPy, di mana setiap baris mewakili satu titik data. Data X selalu diasumsikan sebagai array NumPy atau matriks sparse SciPy yang memiliki entri kontinu (titik mengambang). Algoritme yang diawasi juga memerlukan argumen y, yang merupakan array NumPy satu dimensi yang berisi nilai target untuk regresi atau klasifikasi (yaitu, label atau respons keluaran yang diketahui).

Ada dua cara utama untuk menerapkan model yang dipelajari dalam scikit-learn. Untuk membuat prediksi berupa keluaran baru seperti y, digunakan metode prediksi. Untuk membuat representasi baru dari data input X, Anda menggunakan metode transformasi. Tabel 3-1 merangkum kasus penggunaan metode prediksi dan transformasi.

**Tabel 3.1** ringkasan API scikit-learn

<code>estimator.fit(x_train, [y_train])</code>	
<code>estimator.predict(X_text) estimator.transform(X_test)</code>	
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature extraction
	Feature selection

Selain itu, semua model yang diawasi memiliki metode skor(uji\_x, uji\_y) yang memungkinkan evaluasi model. Pada Tabel 3-1, X\_train dan y\_train mengacu pada data pelatihan dan label pelatihan, sedangkan X\_test dan y\_test mengacu pada data pengujian dan label pengujian (jika berlaku).

## BAB 4

### MEWAKILI FITUR DATA DAN REKAYASA

Sejauh ini, kami berasumsi bahwa data kami masuk sebagai array dua dimensi dari angka floating-point, di mana setiap kolom adalah fitur berkelanjutan yang menggambarkan titik data. Untuk banyak aplikasi, ini bukanlah cara pengumpulan data. Jenis fitur yang sangat umum adalah fitur kategoris. Juga dikenal sebagai fitur diskrit, ini biasanya tidak numerik. Perbedaan antara fitur kategoris dan fitur kontinu analog dengan perbedaan antara klasifikasi dan regresi, hanya pada sisi input daripada sisi output. Contoh fitur kontinu yang telah kita lihat adalah kecerahan piksel dan pengukuran ukuran bunga tanaman. Contoh fitur kategoris adalah merek produk, warna produk, atau departemen (buku, pakaian, perangkat keras) tempat produk itu dijual. Ini semua adalah properti yang dapat menggambarkan suatu produk, tetapi tidak bervariasi dalam cara terus menerus. Sebuah produk milik baik di departemen pakaian atau di departemen buku. Tidak ada jalan tengah antara buku dan pakaian, dan tidak ada tatanan alami untuk kategori yang berbeda (buku tidak lebih besar atau lebih kecil dari pakaian, perangkat keras tidak antara buku dan pakaian, dll.).

Terlepas dari jenis fitur yang terdiri dari data Anda, cara Anda merepresentasikannya dapat memiliki efek yang sangat besar pada performa model pembelajaran mesin. Kita melihat di Bab 2 dan 3 bahwa penskalaan data itu penting. Dengan kata lain, jika Anda tidak mengubah skala data Anda (misalnya, untuk varian unit), maka akan ada perbedaan apakah Anda mewakili pengukuran dalam sentimeter atau inci. Kami juga melihat di Bab 2 bahwa dapat membantu untuk menambah data Anda dengan fitur tambahan, seperti menambahkan interaksi (produk) fitur atau polinomial yang lebih umum.

Pertanyaan tentang cara terbaik untuk merepresentasikan data Anda untuk aplikasi tertentu dikenal sebagai rekayasa fitur, dan itu adalah salah satu tugas utama ilmuwan data dan praktisi pembelajaran mesin yang mencoba memecahkan masalah dunia nyata. Mewakili data Anda dengan cara yang benar dapat memiliki pengaruh yang lebih besar pada performa model yang diawasi daripada parameter persis yang Anda pilih.

Dalam bab ini, pertama-tama kita akan membahas kasus penting dan sangat umum dari fitur kategori, dan kemudian memberikan beberapa contoh transformasi yang berguna untuk kombinasi fitur dan model tertentu.

#### **4.1 VARIABEL KATEGORI**

Sebagai contoh, kami akan menggunakan dataset pendapatan orang dewasa di Amerika Serikat, yang berasal dari database sensus 1994. Tugas kumpulan data dewasa adalah untuk memprediksi apakah seorang pekerja memiliki pendapatan lebih dari \$50.000 atau di bawah \$50.000. Fitur-fitur dalam dataset ini meliputi usia pekerja, cara mereka bekerja (wiraswasta, karyawan industri swasta, pegawai pemerintah, dll.), pendidikan mereka, jenis kelamin mereka, jam kerja per minggu, pekerjaan, dan banyak lagi. Tabel 41 menunjukkan beberapa entri pertama dalam kumpulan data.

**Tabel 4.1** Beberapa entri pertama dalam kumpulan data dewasa

Usia	kelas kerja	pendidikan	jenis kelamin	jam per minggu	pekerjaan	penghasilan
0 39	Pemerintah negara bagian	Sarjana	Pria	40	adm-klerikal	<=50K
1 50	Self-emp-not-inc	Sarjana	Pria	13	eksekutif-manajerial	<=50K
2 38	Pribadi	lulusan HS	Pria	40	Handler-cleaners	<=50K
3 53	Pribadi	tanggal 11	Pria	40	Handler-cleaners	<=50K
4 28	Pribadi	Sarjana	Perempuan	40	Prof-spesialisasi	<=50K
5 37	Pribadi	Master	Perempuan	40	eksekutif-manajerial	<=50K
6 49	Pribadi	tanggal 9	Perempuan	16	Layanan lainnya	<=50K
7 52	Self-emp-not-inc	lulusan HS	Pria	45	eksekutif-manajerial	>50K
8 31	Pribadi	Master	Perempuan	50	Prof-spesialisasi	>50K
9 42	Pribadi	Sarjana	Pria	40	eksekutif-manajerial	>50K
10 37	Pribadi	Beberapa perguruan tinggi	Pria	80	eksekutif-manajerial	>50K

Tugas diutarakan sebagai tugas klasifikasi dengan dua kelas pendapatan <=50rb dan >50rb. Mungkin juga untuk memprediksi pendapatan yang tepat, dan menjadikan ini sebagai tugas regresi. Namun, itu akan jauh lebih sulit, dan divisi 50K menarik untuk dipahami sendiri.

Dalam kumpulan data ini, usia dan jam per minggu adalah fitur berkelanjutan, yang kami tahu cara menanganinya. Namun, fitur kelas kerja, pendidikan, jenis kelamin, dan pekerjaan bersifat kategoris. Semuanya berasal dari daftar nilai yang mungkin, sebagai lawan dari rentang, dan menunjukkan properti kualitatif, sebagai lawan dari kuantitas.

Sebagai titik awal, katakanlah kita ingin mempelajari pengklasifikasi regresi logistik pada data ini. Kita tahu dari Bab 2 bahwa regresi logistik membuat prediksi, , menggunakan rumus berikut:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \dots + w[p] * x[p] + b > 0$$

di mana  $w[i]$  dan  $b$  adalah koefisien yang dipelajari dari set pelatihan dan  $x[i]$  adalah fitur input. Rumus ini masuk akal jika  $x[i]$  adalah angka, tetapi tidak jika  $x[2]$  adalah "Master" atau "Sarjana". Jelas kita perlu merepresentasikan data kita dengan cara yang berbeda ketika menerapkan regresi logistik. Bagian selanjutnya akan menjelaskan bagaimana kita dapat mengatasi masalah ini.

### One-Hot-Encoding (Variabel Dummy)

Sejauh ini cara paling umum untuk merepresentasikan variabel kategori adalah menggunakan pengkodean satu-panas atau pengkodean satu-keluar-N, juga dikenal sebagai variabel dummy. Gagasan di balik variabel dummy adalah mengganti variabel kategori dengan satu atau lebih fitur baru yang dapat memiliki nilai 0 dan 1. Nilai 0 dan 1 masuk akal dalam rumus untuk klasifikasi biner linier (dan untuk semua model lain dalam scikit-learn), dan kami dapat mewakili sejumlah kategori dengan memperkenalkan satu fitur baru per kategori, seperti yang dijelaskan di sini.

Katakanlah untuk fitur kelas kerja kami memiliki kemungkinan nilai "Pegawai Pemerintah", "Pegawai Swasta", "Pekerja Mandiri", dan "Pekerjaan Mandiri dinilai". Untuk mengkodekan empat kemungkinan nilai ini, kami membuat empat fitur baru, yang disebut "Pegawai Pemerintah", "Karyawan Swasta", "Pekerja Mandiri", dan "Pekerjaan Mandiri". Sebuah fitur adalah 1 jika kelas kerja untuk orang ini memiliki nilai yang sesuai dan 0 sebaliknya, jadi tepat satu dari empat fitur baru akan menjadi 1 untuk setiap titik data. Inilah sebabnya mengapa ini disebut pengkodean satu-panas atau satu-keluar-N.

Prinsip tersebut diilustrasikan pada Tabel 4.2. Satu fitur dikodekan menggunakan empat fitur baru. Saat menggunakan data ini dalam algoritme pembelajaran mesin, kami akan menghapus fitur kelas kerja asli dan hanya mempertahankan fitur 0-1.

**Tabel 4.2** Mengkodekan fitur kelas kerja menggunakan enkode satu-panas

kelas kerja	Pegawai pemerintah	Karyawan Swasta	Bekerja sendiri	Wiraswasta Incorporated
Pegawai pemerintah	1	0	0	0
Karyawan Swasta	0	1	0	0
Bekerja sendiri	0	0	1	0
Wiraswasta Incorporated	0	0	0	1



Encoding one-hot yang kami gunakan cukup mirip, tetapi tidak identik, dengan pengkodean dummy yang digunakan dalam statistik. Untuk mempermudah, kami mengkodekan setiap kategori dengan fitur biner yang berbeda. Dalam statistik, adalah umum untuk mengkodekan fitur kategoris dengan  $k$  nilai yang berbeda yang mungkin menjadi fitur  $k-1$  (yang terakhir direpresentasikan sebagai semua nol). Hal ini dilakukan untuk menyederhanakan analisis (lebih teknis, ini akan menghindari kekurangan peringkat matriks data).

Ada dua cara untuk mengonversi data Anda menjadi pengkodean variabel kategorikal, baik menggunakan pandas atau scikit-learn. Pada saat penulisan, menggunakan panda sedikit lebih mudah, jadi mari kita ikuti rute ini. Pertama kita memuat data menggunakan panda dari file comma-separated values (CSV):

In[2]:

```
import pandas as pd
# The file has no headers naming the columns, so we pass header=None
# and provide the column names explicitly in "names"
data = pd.read_csv(
    "/home/andy/datasets/adult.data", header=None, index_col=False,
    names=['age', 'workclass', 'fnlwgt', 'education', 'education-num',
           'marital-status', 'occupation', 'relationship', 'race', 'gender',
           'capital-gain', 'capital-loss', 'hours-per-week', 'native-country',
           'income'])
# For illustration purposes, we only select some of the columns
data = data[['age', 'workclass', 'education', 'gender', 'hours-per-week',
             'occupation', 'income']]
# IPython.display allows nice output formatting within the Jupyter notebook
display(data.head())
```

Tabel 4.3 menunjukkan hasilnya.

**Tabel 4.3** Lima baris pertama dari kumpulan data dewasa

Usia	kelas kerja	pendidikan	jenis kelamin	jam per minggu pekerjaan	penghasilan
0 39	Pemerintah negara bagian	Sarjana	Pria	40	adm-klerikal <=50K
1 50	Self-emp-not-inc	Sarjana	Pria	13	eksekutif-manajerial <=50K
2 38	Pribadi	Iulusan HS	Pria	40	Handler-cleaners <=50K
3 53	Pribadi	tanggal 11	Pria	40	Handler-cleaners <=50K
4 28	Pribadi	Sarjana	Perempuan	40	Prof-specialty <=50K

### Memeriksa data kategorikal yang disandikan dengan string

Setelah membaca kumpulan data seperti ini, sering kali baik untuk memeriksa terlebih dahulu apakah sebuah kolom benar-benar berisi data kategorikal yang berarti. Saat bekerja dengan data yang dimasukkan oleh manusia (misalnya, pengguna di situs web), mungkin tidak ada kumpulan kategori yang tetap, dan perbedaan dalam ejaan dan kapitalisasi mungkin memerlukan pemrosesan awal. Misalnya, mungkin beberapa orang menetapkan jenis kelamin sebagai "pria" dan beberapa sebagai "pria", dan kami mungkin ingin merepresentasikan dua input ini menggunakan kategori yang sama. Cara yang baik untuk memeriksa isi kolom adalah menggunakan fungsi `value_counts` dari Pandas Series (tipe kolom tunggal dalam DataFrame), untuk menunjukkan kepada kita apa nilai unik itu dan seberapa sering mereka muncul:

**In[3]:**

```
print(data.gender.value_counts())
```

**Out[3]:**

Gender	Count
Male	21790
Female	10771
Name: gender, dtype: int64	

Kita dapat melihat bahwa ada tepat dua nilai untuk gender dalam dataset ini, Male dan Female, artinya data tersebut sudah dalam format yang baik untuk direpresentasikan menggunakan *one-hot-encoding*. Dalam aplikasi nyata, Anda harus melihat semua kolom dan memeriksa nilainya. Kami akan melewatkannya di sini untuk singkatnya.

Ada cara yang sangat sederhana untuk mengkodekan data dalam pandas, menggunakan fungsi `get_dummies`. Fungsi `get_dummies` secara otomatis mengubah semua kolom yang memiliki tipe objek (seperti string) atau kategoris (yang merupakan konsep pandas khusus yang belum kita bicarakan):

**In[4]:**

```
print("Original features:\n", list(data.columns), "\n")
data_dummies = pd.get_dummies(data)
print("Features after get_dummies:\n", list(data_dummies.columns))
```

**Out[4]:**

```
Original features:
['age', 'workclass', 'education', 'gender', 'hours-per-week', 'occupation',
 'income']

Features after get_dummies:
['age', 'hours-per-week', 'workclass_ ?', 'workclass_Federal-gov',
 'workclass_Local-gov', 'workclass_Never-worked', 'workclass_Private',
 'workclass_Self-emp-inc', 'workclass_Self-emp-not-inc',
 'workclass_State-gov', 'workclass_Without-pay', 'education_10th',
 'education_11th', 'education_12th', 'education_1st-4th',
 ...
 'education_Preschool', 'education_Prof-school', 'education_Some-college',
 'gender_Female', 'gender_Male', 'occupation_ ?',
 'occupation_Adm-clerical', 'occupation_Armed-Forces',
 'occupation_Craft-repair', 'occupation_Exec-managerial',
 'occupation_Farming-fishing', 'occupation_Handlers-cleaners',
 ...
 'occupation_Tech-support', 'occupation_Transport-moving',
 'income_<=50K', 'income_>50K']
```

Anda dapat melihat bahwa fitur berkelanjutan usia dan jam per minggu tidak disentuh, sedangkan fitur kategoris diperluas menjadi satu fitur baru untuk setiap nilai yang mungkin:

**In[5]:**

```
data_dummies.head()
```

**Out[5]:**

**Tabel 4.3 Lanjutan**

usia	jam per minggu	kelas kerja_?	kelas kerja_Federal-pemerintah	kelas kerja_lokal	... Pemerintah lokal	pekerjaan_Dukungan teknis	pekerjaan_Transportasi-pindah	penghasilan_=50K	penghasilan_=50K
0 39	40	0.0	0.0	0.0	... 0.0	0.0	0.0	1.0	0.0
1 50	13	0.0	0.0	0.0	... 0.0	0.0	0.0	1.0	0.0
2 38	40	0.0	0.0	0.0	... 0.0	0.0	0.0	1.0	0.0
3 53	40	0.0	0.0	0.0	... 0.0	0.0	0.0	1.0	0.0
4 28	40	0.0	0.0	0.0	... 0.0	0.0	0.0	1.0	0.0

Kita sekarang dapat menggunakan atribut `values` untuk mengonversi `DataFrame` menjadi array NumPy, dan kemudian melatih model pembelajaran mesin di atasnya. Berhati-hatilah untuk memisahkan variabel target (yang sekarang dikodekan dalam dua kolom pendapatan) dari data sebelum melatih model. Menyertakan variabel keluaran, atau beberapa properti turunan dari variabel keluaran, ke dalam representasi fitur adalah kesalahan yang sangat umum dalam membangun model pembelajaran mesin yang diawasi.



Hati-hati: pengindeksan kolom di pandas menyertakan akhir rentang, jadi `'age':'occupation_Transport-moving'` sudah termasuk `pekerjaan_Transport-moving`. Ini berbeda dengan mengiris array NumPy, di mana akhir rentang tidak disertakan: misalnya, `np.arange(11)[0:10]` tidak menyertakan entri dengan indeks 10.

Dalam hal ini, kami hanya mengekstrak kolom yang berisi fitur—yaitu, semua kolom dari `usia` hingga `pekerjaan_Pengangkutan-pindah`. Rentang ini berisi semua fitur tetapi bukan targetnya:

**In[6]:**

```
features = data_dummies.ix[:, 'age':'occupation_Transport-moving']
# Extract NumPy arrays
X = features.values
y = data_dummies['income_>50K'].values
print("X.shape: {} y.shape: {}".format(X.shape, y.shape))
```

**Out[6]:**

`X.shape: (32561, 44) y.shape: (32561,)`

Sekarang data direpresentasikan dengan cara yang dapat digunakan scikit-learn, dan kita dapat melanjutkan seperti biasa:

In[7]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
print("Test score: {:.2f}".format(logreg.score(X_test, y_test)))
```

Out[7]:

Test score: 0.81



Dalam contoh ini, kami memanggil `get_dummies` pada DataFrame yang berisi data pelatihan dan pengujian. Hal ini penting untuk memastikan nilai kategoris direpresentasikan dengan cara yang sama dalam set pelatihan dan set pengujian.

Bayangkan kita memiliki set pelatihan dan pengujian dalam dua Bingkai Data yang berbeda. Jika nilai "Karyawan Pribadi" untuk fitur kelas kerja tidak muncul di set pengujian, panda akan menganggap hanya ada tiga kemungkinan nilai untuk fitur ini dan hanya akan membuat tiga fitur tiruan baru. Sekarang set pelatihan dan pengujian kami memiliki jumlah fitur yang berbeda, dan kami tidak dapat menerapkan model yang kami pelajari di set pelatihan ke set pengujian lagi. Lebih buruk lagi, bayangkan fitur kelas kerja memiliki nilai "Pegawai Pemerintah" dan "Pegawai Swasta" di set pelatihan, dan "Pekerja Mandiri" dan "Pekerja Mandiri Incorporated" di set tes. Dalam kedua kasus, panda akan membuat dua fitur tiruan baru, sehingga Bingkai Data yang disandikan akan memiliki jumlah fitur yang sama. Namun, dua fitur dummy memiliki arti yang sama sekali berbeda dalam set pelatihan dan tes. Kolom yang berarti "Pegawai Pemerintah" untuk set pelatihan akan mengkodekan "Wiraswasta" untuk set tes.

Jika kami membangun model pembelajaran mesin pada data ini, itu akan bekerja sangat buruk, karena akan menganggap kolom memiliki arti yang sama (karena mereka berada di posisi yang sama) padahal sebenarnya mereka memiliki arti yang sangat berbeda. Untuk memperbaikinya, panggil `get_dummies` pada DataFrame yang berisi titik data pelatihan dan pengujian, atau pastikan nama kolom sama untuk set pelatihan dan pengujian setelah memanggil `get_dummies`, untuk memastikan keduanya memiliki semantik yang sama.

## 4.2 ANGKA DAPAT MENGODEKAN KATEGORIS

Dalam contoh kumpulan data dewasa, variabel kategorikal dikodekan sebagai string. Di satu sisi, itu membuka kemungkinan kesalahan ejaan, tetapi di sisi lain, ini dengan jelas menandai variabel sebagai kategoris. Seringkali, baik untuk kemudahan penyimpanan atau karena cara data dikumpulkan, variabel kategori dikodekan sebagai bilangan bulat. Sebagai contoh, bayangkan data sensus dalam dataset dewasa dikumpulkan menggunakan kuesioner, dan jawaban untuk kelas kerja dicatat sebagai 0 (kotak pertama dicentang), 1 (kotak kedua dicentang), 2 (kotak ketiga dicentang), dan seterusnya. Sekarang kolom akan berisi angka dari 0 hingga 8, alih-alih string seperti "Pribadi", dan tidak akan langsung terlihat jelas bagi seseorang yang melihat tabel yang mewakili kumpulan data apakah mereka harus

memperlakukan variabel ini sebagai kontinu atau kategoris. Mengetahui bahwa angka-angka tersebut menunjukkan status pekerjaan, bagaimanapun, jelas bahwa ini adalah negara bagian yang sangat berbeda dan tidak boleh dimodelkan oleh satu variabel kontinu.



Fitur kategoris sering dikodekan menggunakan bilangan bulat. Bahwa mereka adalah angka tidak berarti bahwa mereka harus diperlakukan sebagai fitur berkelanjutan. Tidak selalu jelas apakah fitur integer harus diperlakukan sebagai kontinu atau diskrit (dan dikodekan satu-panas). Jika tidak ada urutan antara semantik yang dikodekan (seperti dalam contoh kelas kerja), fitur harus diperlakukan sebagai diskrit. Untuk kasus lain, seperti peringkat bintang lima, pengkodean yang lebih baik bergantung pada tugas dan data tertentu dan algoritme pembelajaran mesin mana yang digunakan.

Fungsi `get_dummies` di pandas memperlakukan semua angka sebagai kontinu dan tidak akan membuat variabel dummy untuk mereka. Untuk menyiasatinya, Anda dapat menggunakan `OneHotEncoder` scikit-learning, yang dapat digunakan untuk menentukan variabel mana yang kontinu dan mana yang diskrit, atau mengonversi kolom numerik di DataFrame menjadi string. Sebagai ilustrasi, mari buat objek DataFrame dengan dua kolom, satu berisi string dan satu berisi bilangan bulat:

**In[8]:**

```
# create a DataFrame with an integer feature and a categorical string feature
demo_df = pd.DataFrame({'Integer Feature': [0, 1, 2, 1],
                        'Categorical Feature': ['socks', 'fox', 'socks', 'box']})
display(demo_df)
```

Tabel 4.4 menunjukkan hasilnya.

**Tabel 4.4** DataFrame berisi fitur string kategoris dan fitur integer

Fitur Kategoris	Fitur Bilangan Bulat
0 socks	0
1 fox	1
2 socks	2
3 box	1

Menggunakan `get_dummies` hanya akan mengkodekan fitur string dan tidak akan mengubah fitur integer, seperti yang Anda lihat pada Tabel 4.5:

**In[9]:**

```
pd.get_dummies(demo_df)
```

**Tabel 4.5** Versi data yang disandikan satu-panas dari **Tabel 4.4**, membiarkan fitur bilangan bulat tidak berubah

Fitur Bilangan Bulat	Feature_box	Fitur Kategoris_fox	Feature_socks

0 0	0.0	0.0	1.0
1 1	0.0	1.0	0.0
2 2	0.0	0.0	1.0
3 1	1.0	0.0	0.0

Jika Anda ingin variabel dummy dibuat untuk kolom “Fitur Bilangan Bulat”, Anda dapat secara eksplisit mencantumkan kolom yang ingin Anda enkode menggunakan parameter `columns`. Kemudian, kedua fitur tersebut akan diperlakukan sebagai kategoris (lihat Tabel 4.6):

**In[10]:**

```
demo_df['Integer Feature'] = demo_df['Integer Feature'].astype(str)
pd.get_dummies(demo_df, columns=['Integer Feature', 'Categorical Feature'])
```

**Tabel 4.6** Encoding *one-hot* dari data yang ditunjukkan pada Tabel 4.4, mengkodekan fitur integer dan string

Fitur Bilangan Bulat_0	Fitur Bilangan Bulat_1	Fitur Bilangan Bulat_2	Feature_box kategoris	Fitur Kategoris_fo x	Feature_socks kategoris
0 1.0	0.0	0.0	0.0	0.0	1.0
1 0.0	1.0	0.0	0.0	1.0	0.0
2 0.0	0.0	1.0	0.0	0.0	1.0
3 0.0	1.0	0.0	1.0	0.0	0.0

### 4.3 BINNING, DISKRITISASI, MODEL LINIER, DAN POHON

Cara terbaik untuk merepresentasikan data tidak hanya bergantung pada semantik data, tetapi juga pada jenis model yang Anda gunakan. Model linier dan model berbasis pohon (seperti pohon keputusan, pohon yang didorong gradien, dan hutan acak), dua keluarga besar dan sangat umum digunakan, memiliki sifat yang sangat berbeda dalam hal bagaimana mereka bekerja dengan representasi fitur yang berbeda. Mari kembali ke kumpulan data regresi gelombang yang kita gunakan di Bab 2. Ini hanya memiliki satu fitur input.

Berikut adalah perbandingan model regresi linier dan regresi pohon keputusan pada dataset ini (lihat Gambar 4.1):

In[11]:

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

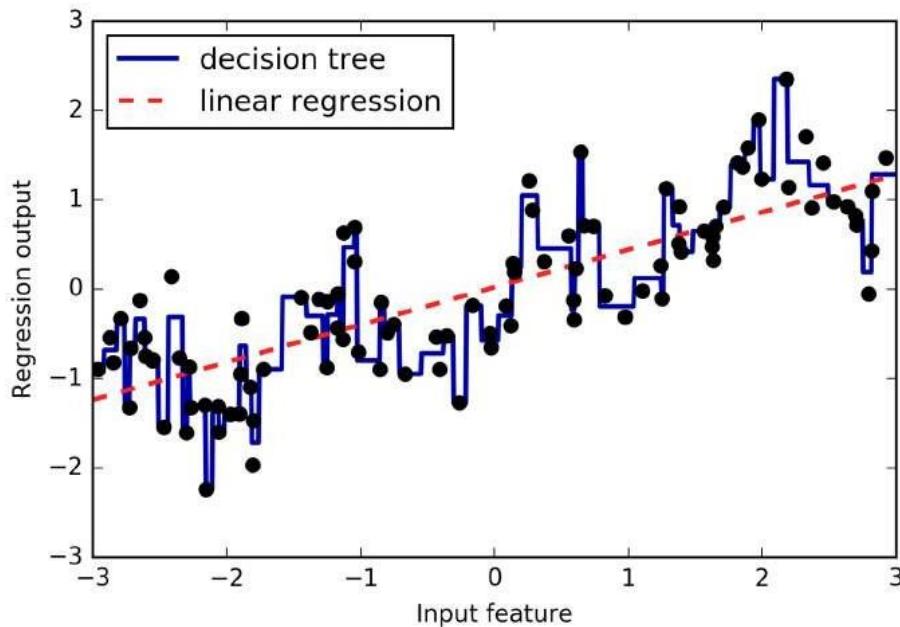
X, y = mglearn.datasets.make_wave(n_samples=100)
line = np.linspace(-3, 3, 1000, endpoint=False).reshape(-1, 1)

reg = DecisionTreeRegressor(min_samples_split=3).fit(X, y)
plt.plot(line, reg.predict(line), label="decision tree")

reg = LinearRegression().fit(X, y)
plt.plot(line, reg.predict(line), label="linear regression")

plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")
```

Seperti yang Anda ketahui, model linier hanya dapat memodelkan hubungan linier, yang merupakan garis dalam kasus fitur tunggal. Pohon keputusan dapat membangun model data yang jauh lebih kompleks. Namun, ini sangat tergantung pada representasi data. Salah satu cara untuk membuat model linier lebih kuat pada data kontinu adalah dengan menggunakan binning (juga dikenal sebagai diskritisasi) fitur untuk membaginya menjadi beberapa fitur, seperti yang dijelaskan di sini.



**Gambar 4.1** Membandingkan regresi linier dan pohon keputusan pada kumpulan data gelombang

Kami membayangkan sebuah partisi dari rentang input untuk fitur (dalam hal ini, angka dari  $-3$  hingga  $3$ ) ke dalam jumlah bin yang tetap—katakanlah,  $10$ . Sebuah titik data kemudian akan diwakili oleh bin mana ia jatuh. Untuk menentukan ini, pertama-tama kita harus mendefinisikan bin. Dalam hal ini, kami akan mendefinisikan  $10$  nampan dengan jarak yang sama antara  $-3$  dan  $3$ . Kami menggunakan fungsi `np.linspace` untuk ini, membuat  $11$  entri, yang akan membuat  $10$  nampan—ini adalah spasi di antara dua batas berurutan:

**In[12]:**

```
bins = np.linspace(-3, 3, 11)
print("bins: {}".format(bins))
```

**Out[12]:**

```
bins: [-3. -2.4 -1.8 -1.2 -0.6  0.   0.6  1.2  1.8  2.4  3. ]
```

Di sini, nampang pertama berisi semua titik data dengan nilai fitur -3 hingga -2,68, nampang kedua berisi semua titik dengan nilai fitur dari -2,68 hingga -2,37, dan seterusnya.

Selanjutnya, kami merekam untuk setiap titik data tempat sampah itu berada. Ini dapat dengan mudah dihitung menggunakan fungsi np.digitize:

**In[13]:**

```
which_bin = np.digitize(X, bins=bins)
print("\nData points:\n", X[:5])
print("\nBin membership for data points:\n", which_bin[:5])
```

**Out[13]:**

```
Data points:
[[-0.753]
 [ 2.704]
 [ 1.392]
 [ 0.592]
 [-2.064]]

Bin membership for data points:
[[ 4]
 [10]
 [ 8]
 [ 6]
 [ 2]]
```

Apa yang kami lakukan di sini adalah mengubah fitur input kontinu tunggal dalam kumpulan data gelombang menjadi fitur kategoris yang mengkodekan bin mana titik data berada. Untuk menggunakan model scikit-belajar pada data ini, kami mengubah fitur diskrit ini menjadi one-hot encoding menggunakan OneHotEncoder dari modul preprocessing. OneHotEncoder melakukan pengkodean yang sama seperti pandas.get\_dummies, meskipun saat ini hanya bekerja pada variabel kategori yang merupakan bilangan bulat:

**In[14]:**

```
from sklearn.preprocessing import OneHotEncoder
# transform using the OneHotEncoder
encoder = OneHotEncoder(sparse=False)
# encoder.fit finds the unique values that appear in which_bin
encoder.fit(which_bin)
# transform creates the one-hot encoding
X_binned = encoder.transform(which_bin)
print(X_binned[:5])
```

**Out[14]:**

```
[[ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

Karena kami menentukan 10 bin, dataset yang diubah X\_binned sekarang terdiri dari 10 fitur:

**In[15]:**

```
print("X_binned.shape: {}".format(X_binned.shape))
```

**Out[15]:**

```
X_binned.shape: (100, 10)
```

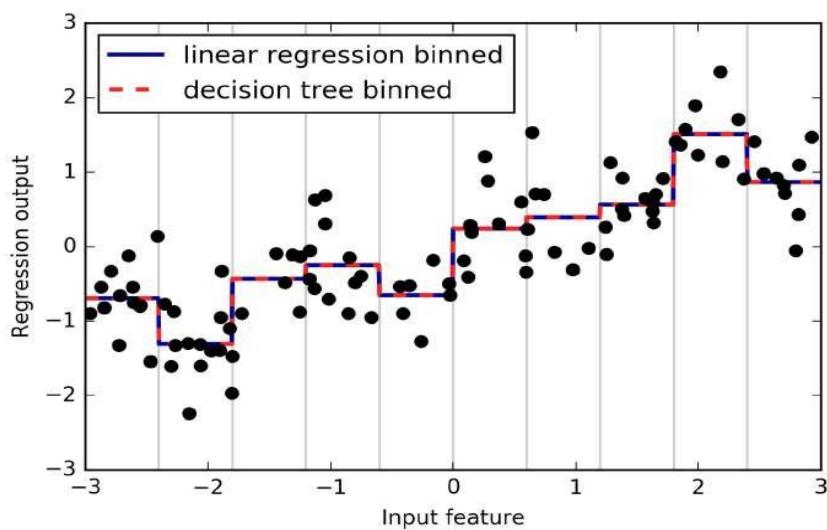
Sekarang kita membangun model regresi linier baru dan model pohon keputusan baru pada data one-hot-encoded. Hasilnya divisualisasikan pada Gambar 4.2, bersama dengan batas bin, ditunjukkan sebagai garis hitam putus-putus:

**In[16]:**

```
line_binned = encoder.transform(np.digitize(line, bins=bins))

reg = LinearRegression().fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='linear regression binned')

reg = DecisionTreeRegressor(min_samples_split=3).fit(X_binned, y)
plt.plot(line, reg.predict(line_binned), label='decision tree binned')
plt.plot(X[:, 0], y, 'o', c='k')
plt.vlines(bins, -3, 3, linewidth=1, alpha=.2)
plt.legend(loc="best")
plt.ylabel("Regression output")
plt.xlabel("Input feature")
```



**Gambar 4.2** Membandingkan regresi linier dan regresi pohon keputusan pada fitur binned

Garis putus-putus dan garis padat berada tepat di atas satu sama lain, artinya model regresi linier dan pohon keputusan membuat prediksi yang persis sama. Untuk setiap bin, mereka memprediksi nilai konstan. Karena fitur konstan dalam setiap bin, model apapun harus memprediksi nilai yang sama untuk semua titik dalam bin. Membandingkan apa yang

dipelajari model sebelum menggabungkan fitur dan setelahnya, kita melihat bahwa model linier menjadi jauh lebih fleksibel, karena sekarang memiliki nilai yang berbeda untuk setiap bin, sedangkan model pohon keputusan menjadi kurang fleksibel. Fitur binning umumnya tidak memiliki efek yang menguntungkan untuk model berbasis pohon, karena model ini dapat belajar untuk memisahkan data di mana saja. Dalam arti tertentu, itu berarti pohon keputusan dapat mempelajari binning apa pun yang paling berguna untuk memprediksi data ini. Selain itu, pohon keputusan melihat beberapa fitur sekaligus, sementara binning biasanya dilakukan per fitur. Namun, model linier sangat diuntungkan dalam ekspresif dari transformasi data.

Jika ada alasan bagus untuk menggunakan model linier untuk kumpulan data tertentu—misalnya, karena sangat besar dan berdimensi tinggi, tetapi beberapa fitur memiliki hubungan nonlinier dengan output—binning bisa menjadi cara yang bagus untuk meningkatkan kekuatan pemodelan.

#### 4.4 INTERAKSI DAN POLINOMIAL

Cara lain untuk memperkaya representasi fitur, terutama untuk model linier, adalah menambahkan fitur interaksi dan fitur polinomial dari data asli. Rekayasa fitur semacam ini sering digunakan dalam pemodelan statistik, tetapi juga umum di banyak aplikasi pembelajaran mesin praktis.

Sebagai contoh pertama, lihat kembali Gambar 4.2. Model linier mempelajari nilai konstan untuk setiap bin dalam kumpulan data gelombang. Namun, kita tahu bahwa model linier tidak hanya dapat mempelajari offset, tetapi juga kemiringan. Salah satu cara untuk menambahkan kemiringan ke model linier pada data binned adalah dengan menambahkan fitur asli (sumbu x dalam plot) kembali. Ini mengarah ke kumpulan data 11-dimensi, seperti yang terlihat pada Gambar 4.3:

**In[17]:**

```
X_combined = np.hstack([X, X_binned])
print(X_combined.shape)
```

**Out[17]:**

```
(100, 11)
```

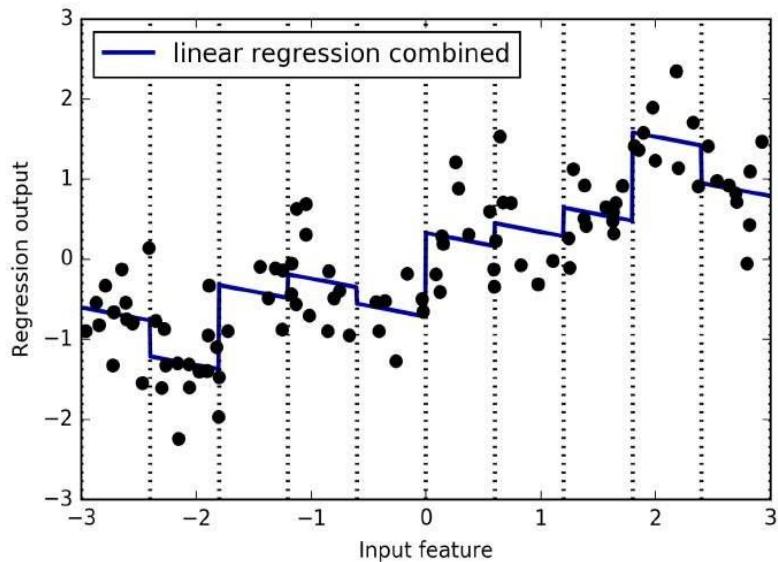
**In[18]:**

```
reg = LinearRegression().fit(X_combined, y)

line_combined = np.hstack([line, line_binned])
plt.plot(line, reg.predict(line_combined), label='linear regression combined')

for bin in bins:
    plt.plot([bin, bin], [-3, 3], ':', c='k')

plt.legend(loc="best")
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.plot(X[:, 0], y, 'o', c='k')
```



**Gambar 4.3** Regresi linier menggunakan fitur binned dan kemiringan global tunggal

Dalam contoh ini, model mempelajari offset untuk setiap nampan, bersama dengan kemiringan. Kemiringan yang dipelajari adalah ke bawah, dan dibagi di semua wadah—ada fitur sumbu-x tunggal, yang memiliki kemiringan tunggal. Karena kemiringan dibagi di semua tempat sampah, tampaknya tidak terlalu membantu. Kami lebih suka memiliki kemiringan terpisah untuk setiap tempat sampah! Kita dapat mencapai ini dengan menambahkan interaksi atau fitur produk yang menunjukkan bin mana titik data berada dan di mana letaknya pada sumbu x. Fitur ini adalah produk dari indikator nampan dan fitur asli. Mari kita buat kumpulan data ini:

**In[19]:**

```
X_product = np.hstack([X_binned, X * X_binned])
print(X_product.shape)
```

**Out[19]:**

```
(100, 20)
```

Dataset sekarang memiliki 20 fitur: indikator tempat titik data berada, dan produk dari fitur asli dan indikator nampan. Anda dapat menganggap fitur produk sebagai salinan terpisah dari fitur sumbu x untuk setiap nampan. Ini adalah fitur asli di dalam tempat sampah, dan nol di tempat lain. Gambar 4.4 menunjukkan hasil model linier pada representasi baru ini:

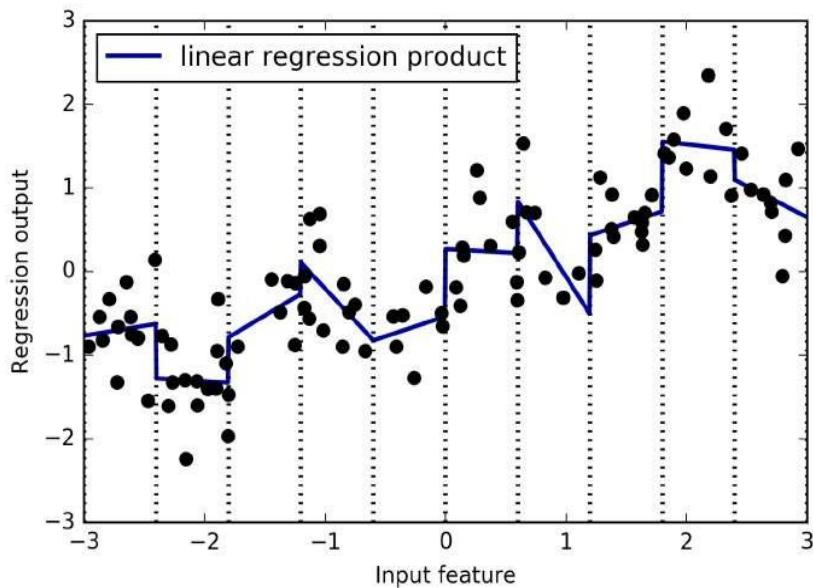
**In[20]:**

```
reg = LinearRegression().fit(X_product, y)

line_product = np.hstack([line_binned, line * line_binned])
plt.plot(line, reg.predict(line_product), label='linear regression product')

for bin in bins:
    plt.plot([bin, bin], [-3, 3], ':', c='k')

plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")
```



**Gambar 4.4** Regresi linier dengan kemiringan terpisah per bin

Seperti yang Anda lihat, sekarang setiap nampan memiliki offset dan kemiringan sendiri dalam model ini.

Menggunakan binning adalah salah satu cara untuk memperluas fitur berkelanjutan. Satu lagi adalah dengan menggunakan polinomial dari fitur asli. Untuk fitur  $x$  yang diberikan, kita mungkin ingin mempertimbangkan  $x^{**} 2$ ,  $x^{**} 3$ ,  $x^{**} 4$ , dan seterusnya. Ini diimplementasikan dalam `PolynomialFeatures` dalam modul `preprocessing`:

**In[21]:**

```
from sklearn.preprocessing import PolynomialFeatures

# include polynomials up to x ** 10:
# the default "include_bias=True" adds a feature that's constantly 1
poly = PolynomialFeatures(degree=10, include_bias=False)
poly.fit(X)
X_poly = poly.transform(X)
```

Menggunakan derajat 10 menghasilkan 10 fitur:

**In[22]:**

```
print("X_poly.shape: {}".format(X_poly.shape))
```

**Out[22]:**

```
X_poly.shape: (100, 10)
```

Mari kita bandingkan entri `X_poly` dengan entri `X`:

**In[23]:**

```
print("Entries of X:\n{}".format(X[:5]))
print("Entries of X_poly:\n{}".format(X_poly[:5]))
```

**Out[23]:**

```

Entries of X:
[[ -0.753]
 [ 2.704]
 [ 1.392]
 [ 0.592]
 [-2.064]]
Entries of X_poly:
[[ -0.753      0.567     -0.427      0.321     -0.242      0.182
   -0.137      0.103     -0.078      0.058]
 [ 2.704      7.313    19.777     53.482    144.632    391.125
  1057.714   2860.360  7735.232  20918.278]
 [ 1.392      1.938     2.697      3.754      5.226      7.274
  10.125     14.094    19.618    27.307]
 [ 0.592      0.350     0.207      0.123      0.073      0.043
  0.025      0.015     0.009      0.005]
 [-2.064      4.260     -8.791     18.144    -37.448     77.289
 -159.516    329.222   -679.478   1402.367]]
```

Anda dapat memperoleh semantik fitur dengan memanggil `get_feature_names` metode, yang menyediakan eksponen untuk setiap fitur:

**In[24]:**

```
print("Polynomial feature names:\n{}".format(poly.get_feature_names()))
```

**Out[24]:**

```

Polynomial feature names:
['x0', 'x0^2', 'x0^3', 'x0^4', 'x0^5', 'x0^6', 'x0^7', 'x0^8', 'x0^9', 'x0^10']
```

Anda dapat melihat bahwa kolom pertama `X_poly` sama persis dengan `X`, sedangkan kolom lainnya adalah pangkat dari entri pertama. Sangat menarik untuk melihat seberapa besar beberapa nilai yang bisa didapat. Kolom kedua memiliki entri di atas 20.000, urutan besarnya berbeda dari yang lain.

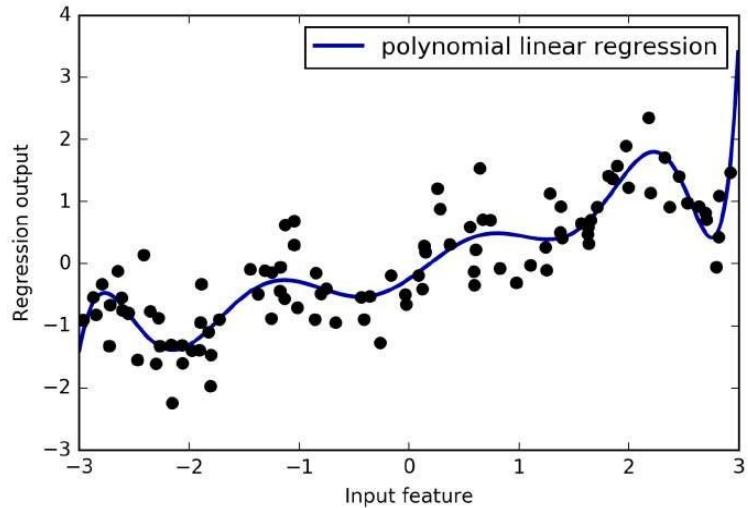
Menggunakan fitur polinomial bersama-sama dengan model regresi linier menghasilkan model klasik regresi polinomial (lihat Gambar 4.5):

**In[26]:**

```

reg = LinearRegression().fit(X_poly, y)

line_poly = poly.transform(line)
plt.plot(line, reg.predict(line_poly), label='polynomial linear regression')
plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")
```



**Gambar 4.5** Regresi linier dengan fitur polinomial derajat kesepuluh

Seperti yang Anda lihat, fitur polinomial menghasilkan kecocokan yang sangat halus pada data satu dimensi ini. Namun, polinomial derajat tinggi cenderung berperilaku ekstrim pada batas atau di daerah dengan sedikit data.

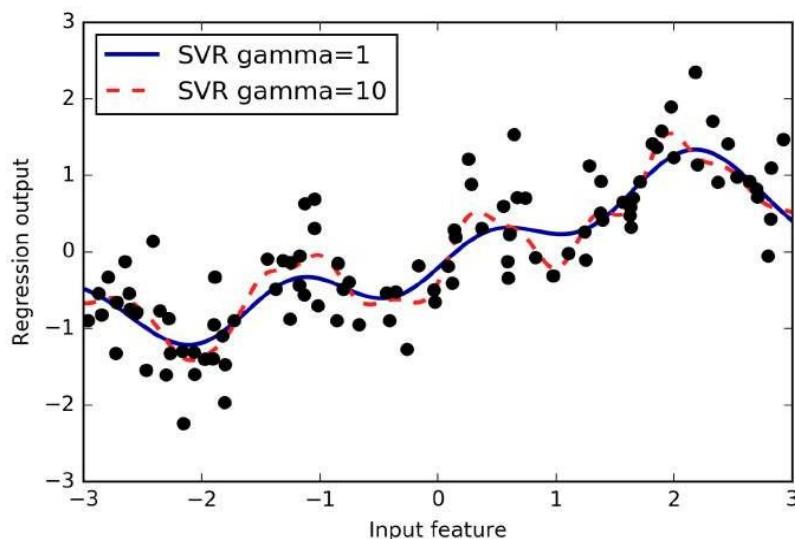
Sebagai perbandingan, berikut adalah model SVM kernel yang dipelajari pada data asli, tanpa transformasi apa pun (lihat Gambar 4.6):

In[26]:

```
from sklearn.svm import SVR

for gamma in [1, 10]:
    svr = SVR(gamma=gamma).fit(X, y)
    plt.plot(line, svr.predict(line), label='SVR gamma={}'.format(gamma))

plt.plot(X[:, 0], y, 'o', c='k')
plt.ylabel("Regression output")
plt.xlabel("Input feature")
plt.legend(loc="best")
```



**Gambar 4.6** Perbandingan parameter gamma yang berbeda untuk SVM dengan kernel RBF

Dengan menggunakan model yang lebih kompleks, kernel SVM, kita dapat mempelajari prediksi kompleks yang serupa dengan regresi polinomial tanpa transformasi fitur yang eksplisit.

Sebagai aplikasi interaksi dan polinomial yang lebih realistik, mari kita lihat kembali kumpulan data Boston Housing. Kami telah menggunakan fitur polinomial pada kumpulan data ini di Bab 2. Sekarang mari kita lihat bagaimana fitur ini dibangun, dan seberapa banyak fitur polinomial membantu. Pertama kita memuat data, dan mengubah skalanya menjadi antara 0 dan 1 menggunakan MinMaxScaler:

**In[27]:**

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split
    (boston.data, boston.target, random_state=0)

# rescale data
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Sekarang, kami mengekstrak fitur dan interaksi polinomial hingga tingkat 2:

**In[28]:**

```
poly = PolynomialFeatures(degree=2).fit(X_train_scaled)
X_train_poly = poly.transform(X_train_scaled)
X_test_poly = poly.transform(X_test_scaled)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_poly.shape: {}".format(X_train_poly.shape))
```

**Out[28]:**

```
X_train.shape: (379, 13)
X_train_poly.shape: (379, 105)
```

Data awalnya memiliki 13 fitur, yang diperluas menjadi 105 fitur interaksi. Fitur baru ini mewakili semua kemungkinan interaksi antara dua fitur asli yang berbeda, serta kuadrat dari setiap fitur asli. degree=2 di sini berarti kita melihat semua fitur yang merupakan produk dari hingga dua fitur asli. Korespondensi yang tepat antara fitur input dan output dapat ditemukan menggunakan metode get\_feature\_names :

**In[29]:**

```
print("Polynomial feature names:\n{}".format(poly.get_feature_names()))
```

**Out[29]:**

```
Polynomial feature names:
['1', 'x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10',
 'x11', 'x12', 'x0^2', 'x0 x1', 'x0 x2', 'x0 x3', 'x0 x4', 'x0 x5', 'x0 x6',
 'x0 x7', 'x0 x8', 'x0 x9', 'x0 x10', 'x0 x11', 'x0 x12', 'x1^2', 'x1 x2',
```

```
'x1 x3', 'x1 x4', 'x1 x5', 'x1 x6', 'x1 x7', 'x1 x8', 'x1 x9', 'x1 x10',
'x1 x11', 'x1 x12', 'x2^2', 'x2 x3', 'x2 x4', 'x2 x5', 'x2 x6', 'x2 x7',
'x2 x8', 'x2 x9', 'x2 x10', 'x2 x11', 'x2 x12', 'x3^2', 'x3 x4', 'x3 x5',
'x3 x6', 'x3 x7', 'x3 x8', 'x3 x9', 'x3 x10', 'x3 x11', 'x3 x12', 'x4^2',
'x4 x5', 'x4 x6', 'x4 x7', 'x4 x8', 'x4 x9', 'x4 x10', 'x4 x11', 'x4 x12',
'x5^2', 'x5 x6', 'x5 x7', 'x5 x8', 'x5 x9', 'x5 x10', 'x5 x11', 'x5 x12',
'x6^2', 'x6 x7', 'x6 x8', 'x6 x9', 'x6 x10', 'x6 x11', 'x6 x12', 'x7^2',
'x7 x8', 'x7 x9', 'x7 x10', 'x7 x11', 'x7 x12', 'x8^2', 'x8 x9', 'x8 x10',
'x8 x11', 'x8 x12', 'x9^2', 'x9 x10', 'x9 x11', 'x9 x12', 'x10^2', 'x10 x11',
'x10 x12', 'x11^2', 'x11 x12', 'x12^2']
```

Fitur baru pertama adalah fitur konstan, yang disebut "1" di sini. 13 fitur berikutnya adalah fitur asli (disebut "x0" hingga "x12"). Kemudian ikuti kuadrat fitur pertama ("x0^2") dan kombinasi fitur pertama dan fitur lainnya.

Mari kita bandingkan kinerja menggunakan Ridge pada data dengan dan tanpa interaksi:

**In[30]:**

```
from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train_scaled, y_train)
print("Score without interactions: {:.3f}".format(
    ridge.score(X_test_scaled, y_test)))
ridge = Ridge().fit(X_train_poly, y_train)
print("Score with interactions: {:.3f}".format(
    ridge.score(X_test_poly, y_test)))
```

**Out[30]:**

```
Score without interactions: 0.621
Score with interactions: 0.753
```

Jelas, interaksi dan fitur polinomial memberi kami peningkatan kinerja yang baik saat menggunakan Ridge. Saat menggunakan model yang lebih kompleks seperti hutan acak, ceritanya sedikit berbeda:

**In[31]:**

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100).fit(X_train_scaled, y_train)
print("Score without interactions: {:.3f}".format(
    rf.score(X_test_scaled, y_test)))
rf = RandomForestRegressor(n_estimators=100).fit(X_train_poly, y_train)
print("Score with interactions: {:.3f}".format(rf.score(X_test_poly, y_test)))
```

**Out[31]:**

```
Score without interactions: 0.799
Score with interactions: 0.763
```

Anda dapat melihat bahwa bahkan tanpa fitur tambahan, hutan acak mengalahkan kinerja Ridge. Menambahkan interaksi dan polinomial sebenarnya sedikit menurunkan kinerja.

## 4.5 TRANSFORMASI NONLINIER UNIVARIAT

Kami baru saja melihat bahwa menambahkan fitur kuadrat atau pangkat tiga dapat membantu model linier untuk regresi. Ada transformasi lain yang sering terbukti berguna

untuk mengubah fitur tertentu: khususnya, menerapkan fungsi matematika seperti log, exp, atau sin. Sementara model berbasis pohon hanya peduli pada urutan fitur, model linier dan jaringan saraf sangat terikat dengan skala dan distribusi setiap fitur, dan jika ada hubungan nonlinier antara fitur dan target, itu menjadi sulit untuk dimodelkan. khususnya dalam regresi. Log fungsi dan exp dapat membantu dengan menyesuaikan skala relatif dalam data sehingga dapat ditangkap lebih baik oleh model linier atau jaringan saraf. Kami melihat penerapannya di Bab 2 dengan data harga memori. Fungsi sin dan cos dapat berguna ketika berhadapan dengan data yang mengkodekan pola periodik.

Sebagian besar model bekerja paling baik ketika setiap fitur (dan dalam regresi juga targetnya) terdistribusi Gaussian secara longgar—yaitu, histogram dari setiap fitur harus memiliki sesuatu yang menyerupai bentuk "kurva lonceng" yang sudah dikenal. Menggunakan transformasi seperti log dan exp adalah cara hacky tapi sederhana dan efisien untuk mencapai ini. Kasus yang sangat umum ketika transformasi semacam itu dapat membantu adalah ketika berhadapan dengan data jumlah bilangan bulat. Dengan menghitung data, yang kami maksud adalah fitur seperti "seberapa sering pengguna A masuk?" Hitungan tidak pernah negatif, dan sering mengikuti pola statistik tertentu. Kami menggunakan kumpulan data sintetis jumlah di sini yang memiliki properti serupa dengan yang dapat Anda temukan di alam liar. Semua fitur bernilai integer, sedangkan responsnya berkelanjutan:

**In[32]:**

```
rnd = np.random.RandomState(0)
X_org = rnd.normal(size=(1000, 3))
w = rnd.normal(size=3)

X = rnd.poisson(10 * np.exp(X_org))
y = np.dot(X_org, w)
```

Mari kita lihat 10 entri pertama dari fitur pertama. Semua adalah nilai integer dan positif, tetapi selain itu sulit untuk melihat pola tertentu. Jika kita menghitung kemunculan setiap nilai, distribusi nilai menjadi lebih jelas:

**In[33]:**

```
print("Number of feature appearances:\n{}".format(np.bincount(X[:, 0])))
```

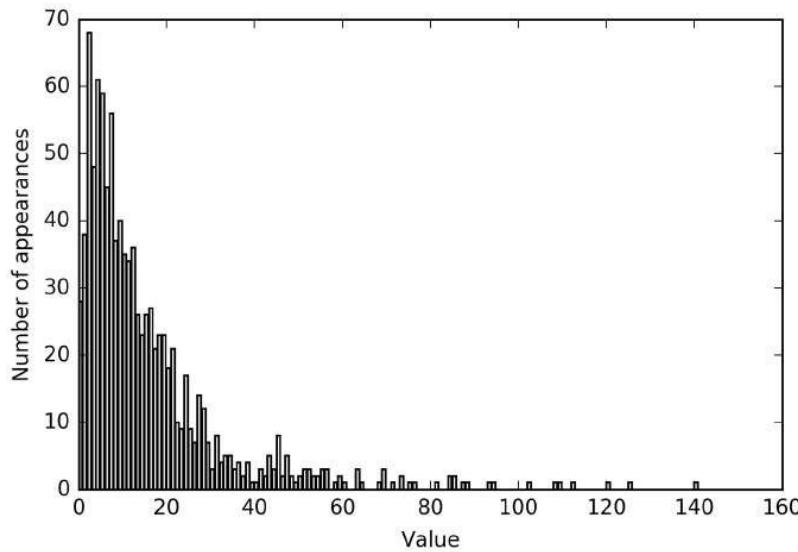
**Out[33]:**

```
Number of feature appearances:
[28 38 68 48 61 59 45 56 37 40 35 34 36 26 23 26 27 21 23 23 18 21 10 9 17
 9 7 14 12 7 3 8 4 5 5 3 4 2 4 1 1 3 2 5 3 8 2 5 2 1
 2 3 3 2 2 3 3 0 1 2 1 0 0 3 1 0 0 0 1 3 0 1 0 2 0
 1 1 0 0 0 0 1 0 0 2 2 0 1 1 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

Nilai 2 tampaknya menjadi yang paling umum, dengan 62 penampilan (bincount selalu dimulai dari 0), dan hitungan untuk nilai yang lebih tinggi turun dengan cepat. Namun, ada beberapa nilai yang sangat tinggi, seperti 134 muncul dua kali. Kami memvisualisasikan hitungan pada Gambar 4.7:

In[34]:

```
bins = np.bincount(X[:, 0])
plt.bar(range(len(bins)), bins, color='w')
plt.ylabel("Number of appearances")
plt.xlabel("Value")
```



Gambar 4.7 Histogram nilai fitur untuk X[0]

Fitur X[:, 1] dan X[:, 2] memiliki properti yang serupa. Distribusi nilai semacam ini (banyak yang kecil dan beberapa yang sangat besar) sangat umum dalam praktik.<sup>1</sup> Namun, ini adalah sesuatu yang kebanyakan model linier tidak dapat tangani dengan baik. Mari kita coba menyesuaikan regresi ridge dengan model ini:

In[35]:

```
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
score = Ridge().fit(X_train, y_train).score(X_test, y_test)
print("Test score: {:.3f}".format(score))
```

Out[35]:

```
Test score: 0.622
```

Seperti yang dapat Anda lihat dari skor R<sup>2</sup> yang relatif rendah, Ridge tidak dapat benar-benar menangkap hubungan antara X dan y. Menerapkan transformasi logaritmik dapat membantu. Karena nilai 0 muncul dalam data (dan logaritma tidak ditentukan pada 0), kita tidak dapat menerapkan log, tetapi kita harus menghitung log(X + 1):

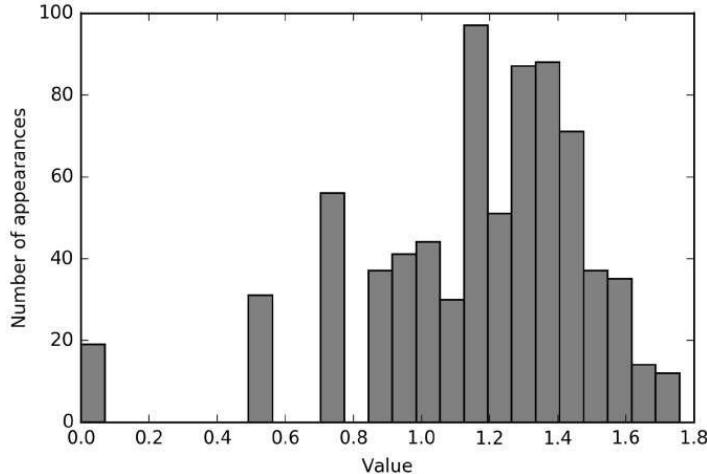
In[36]:

```
X_train_log = np.log(X_train + 1)
X_test_log = np.log(X_test + 1)
```

Setelah transformasi, distribusi data menjadi kurang asimetris dan tidak memiliki outlier yang sangat besar lagi (lihat Gambar 4.8):

In[37]:

```
plt.hist(np.log(X_train_log[:, 0] + 1), bins=25, color='gray')
plt.ylabel("Number of appearances")
plt.xlabel("Value")
```



**Gambar 4.8** Histogram nilai fitur untuk  $X[0]$  setelah transformasi logaritmik

Membangun model punggungan pada data baru memberikan kecocokan yang jauh lebih baik:

In[38]:

```
score = Ridge().fit(X_train_log, y_train).score(X_test_log, y_test)
print("Test score: {:.3f}".format(score))
```

Out[38]:

```
Test score: 0.875
```

Menemukan transformasi yang paling sesuai untuk setiap kombinasi dataset dan model adalah suatu seni. Dalam contoh ini, semua fitur memiliki properti yang sama. Ini jarang terjadi dalam praktik, dan biasanya hanya sebagian dari fitur yang harus diubah, atau terkadang setiap fitur perlu diubah dengan cara yang berbeda. Seperti yang kami sebutkan sebelumnya, jenis transformasi ini tidak relevan untuk model berbasis pohon tetapi mungkin penting untuk model linier. Terkadang juga merupakan ide yang baik untuk mengubah variabel target  $y$  ke dalam regresi. Mencoba memprediksi jumlah (misalnya, jumlah pesanan) adalah tugas yang cukup umum, dan menggunakan transformasi  $\log(y + 1)$  sering membantu.

Seperti yang Anda lihat pada contoh sebelumnya, binning, polinomial, dan interaksi dapat memiliki pengaruh besar pada performa model pada kumpulan data tertentu. Hal ini terutama berlaku untuk model yang tidak terlalu rumit seperti model linier dan model naif Bayes. Model berbasis pohon, di sisi lain, seringkali dapat menemukan interaksi penting itu sendiri, dan tidak memerlukan transformasi data secara eksplisit hampir sepanjang waktu. Model lain, seperti SVM, tetangga terdekat, dan jaringan saraf, kadang-kadang mungkin mendapat manfaat dari penggunaan binning, interaksi, atau polinomial, tetapi implikasinya biasanya jauh lebih tidak jelas daripada dalam kasus model linier.

## 4.6 PEMILIHAN FITUR OTOMATIS

Dengan begitu banyak cara untuk membuat fitur baru, Anda mungkin tergoda untuk meningkatkan dimensi data jauh melampaui jumlah fitur asli. Namun, menambahkan lebih banyak fitur membuat semua model menjadi lebih kompleks, sehingga meningkatkan kemungkinan overfitting. Saat menambahkan fitur baru, atau dengan kumpulan data berdimensi tinggi secara umum, sebaiknya kurangi jumlah fitur menjadi yang paling berguna saja, dan buang sisanya. Hal ini dapat menyebabkan model sederhana yang menggeneralisasi lebih baik. Tapi bagaimana Anda bisa tahu seberapa bagus setiap fiturnya? Ada tiga strategi dasar: statistik univariat, pemilihan berbasis model, dan pemilihan iteratif. Kami akan membahas ketiganya secara rinci. Semua metode tersebut merupakan metode terawasi, artinya membutuhkan target untuk menyesuaikan model. Ini berarti kita perlu membagi data menjadi set pelatihan dan pengujian, dan menyesuaikan pemilihan fitur hanya pada bagian data pelatihan.

## 4.7 STATISTIK UNIVARIAT

Dalam statistik univariat, kami menghitung apakah ada hubungan yang signifikan secara statistik antara setiap fitur dan target. Kemudian fitur-fitur yang terkait dengan kepercayaan tertinggi dipilih. Dalam kasus klasifikasi, ini juga dikenal sebagai analisis varians (ANOVA). Properti kunci dari tes ini adalah bahwa mereka univari- ate, yang berarti bahwa mereka hanya mempertimbangkan setiap fitur secara individual. Akibatnya, sebuah fitur akan dibuang jika hanya informatif bila digabungkan dengan fitur lain. Pengujian univariat seringkali sangat cepat untuk dihitung, dan tidak memerlukan pembuatan model. Di sisi lain, mereka sepenuhnya independen dari model yang mungkin ingin Anda terapkan setelah pemilihan fitur.

Untuk menggunakan pemilihan fitur univariat dalam scikit-learn, Anda perlu memilih tes, biasanya f\_classif (default) untuk klasifikasi atau f\_regression untuk regresi, dan metode untuk membuang fitur berdasarkan nilai-p yang ditentukan dalam pengujian. Semua metode untuk membuang parameter menggunakan ambang batas untuk membuang semua fitur dengan nilai p yang terlalu tinggi (yang berarti mereka tidak mungkin terkait dengan target). Metode berbeda dalam cara mereka menghitung ambang batas ini, dengan yang paling sederhana adalah SelectKBest, yang memilih sejumlah k fitur yang tetap, dan SelectPercentile, yang memilih persentase fitur yang tetap. Mari kita terapkan pemilihan fitur untuk klasifikasi pada dataset kanker. Untuk membuat tugas sedikit lebih sulit, kami akan menambahkan beberapa fitur noise noninformatif ke data. Kami berharap pemilihan fitur dapat mengidentifikasi fitur yang tidak informatif dan menghapusnya:

In[39]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.feature_selection import SelectPercentile
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
```

```
# get deterministic random numbers
rng = np.random.RandomState(42)
noise = rng.normal(size=(len(cancer.data), 50))
# add noise features to the data
# the first 30 features are from the dataset, the next 50 are noise
X_w_noise = np.hstack([cancer.data, noise])

X_train, X_test, y_train, y_test = train_test_split(
    X_w_noise, cancer.target, random_state=0, test_size=.5)
# use f_classif (the default) and SelectPercentile to select 50% of features
select = SelectPercentile(percentile=50)
select.fit(X_train, y_train)
# transform training set
X_train_selected = select.transform(X_train)

print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

**Out[39]:**

```
X_train.shape: (284, 80)
X_train_selected.shape: (284, 40)
```

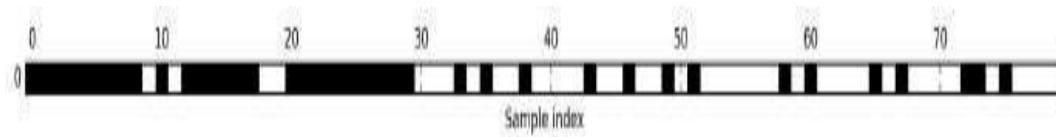
Seperti yang Anda lihat, jumlah fitur dikurangi dari 80 menjadi 40 (50 persen dari jumlah fitur asli). Kita dapat mengetahui fitur mana yang telah dipilih menggunakan metode `get_support`, yang mengembalikan topeng Boolean dari fitur yang dipilih (divisualisasikan pada Gambar 4.9):

**In[40]:**

```
mask = select.get_support()
print(mask)
# visualize the mask -- black is True, white is False
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Sample index")
```

**Out[40]:**

```
[ True  True  True  True  True  True  True  True  True False  True False
 True  True  True  True  True  True False False  True  True  True  True
 True  True  True  True  True  True False False False  True False  True
 False False  True False False False False  True False False  True False
 False  True False  True False False False False False  True False  True
 True False False False  True False  True False  True False False False
 True  True False  True False False False]
```



**Gambar 4.9** Fitur yang dipilih oleh `SelectPercentile`

Seperti yang Anda lihat dari visualisasi topeng, sebagian besar fitur yang dipilih adalah fitur asli, dan sebagian besar fitur noise telah dihapus. Namun, pemulihan fitur asli tidak sempurna. Mari kita bandingkan kinerja regresi logistik pada semua fitur dengan kinerja yang hanya menggunakan fitur yang dipilih:

**In[41]:**

```
from sklearn.linear_model import LogisticRegression

# transform test data
X_test_selected = select.transform(X_test)

lr = LogisticRegression()
lr.fit(X_train, y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test, y_test)))
lr.fit(X_train_selected, y_train)
print("Score with only selected features: {:.3f}".format(
    lr.score(X_test_selected, y_test)))
```

**Out[41]:**

```
Score with all features: 0.930
Score with only selected features: 0.940
```

Dalam hal ini, menghilangkan fitur noise akan meningkatkan kinerja, meskipun beberapa fitur asli hilang. Ini adalah contoh sintetik yang sangat sederhana, dan hasil pada data nyata biasanya bercampur. Pemilihan fitur univariat masih bisa sangat membantu, jika ada begitu banyak fitur sehingga membuat model di atasnya tidak mungkin dilakukan, atau jika Anda menduga bahwa banyak fitur sama sekali tidak informatif.

#### 4.8 PEMILIHAN FITUR BERBASIS MODEL

Pemilihan fitur berbasis model menggunakan model pembelajaran mesin yang diawasi untuk menilai pentingnya setiap fitur, dan hanya menyimpan yang paling penting. Model terawasi yang digunakan untuk pemilihan fitur tidak harus sama dengan model yang digunakan untuk pemodelan terawasi akhir. Model pemilihan fitur perlu memberikan beberapa ukuran kepentingan untuk setiap fitur, sehingga mereka dapat diurutkan berdasarkan ukuran ini. Pohon keputusan dan model berbasis pohon keputusan menyediakan `atribut_fitur_penting`, yang secara langsung mengkodekan pentingnya setiap fitur. Model linier memiliki koefisien, yang juga dapat digunakan untuk menangkap kepentingan fitur dengan mempertimbangkan nilai absolut.

Seperti yang kita lihat di Bab 2, model linier dengan penalti L1 mempelajari koefisien sparse, yang hanya menggunakan sebagian kecil fitur. Ini dapat dilihat sebagai bentuk pemilihan fitur untuk model itu sendiri, tetapi juga dapat digunakan sebagai langkah preprocessing untuk memilih fitur untuk model lain. Berbeda dengan pemilihan univariat, pemilihan berbasis model mempertimbangkan semua fitur sekaligus, sehingga dapat menangkap interaksi (jika model dapat menangkapnya). Untuk menggunakan pemilihan fitur berbasis model, kita perlu menggunakan transformator `SelectFromModel`:

**In[42]:**

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=42),
    threshold="median")
```

Kelas `SelectFromModel` memilih semua fitur yang memiliki ukuran kepentingan fitur (seperti yang disediakan oleh model yang diawasi) lebih besar dari ambang batas yang disediakan. Untuk mendapatkan hasil yang sebanding dengan apa yang kami dapatkan dengan

pemilihan fitur univariat, kami menggunakan median sebagai ambang batas, sehingga setengah dari fitur akan dipilih. Kami menggunakan pengklasifikasi hutan acak dengan 100 pohon untuk menghitung kepentingan fitur. Ini adalah model yang cukup kompleks dan jauh lebih kuat daripada menggunakan tes univariat. Sekarang mari kita benar-benar sesuai dengan modelnya:

**In[43]:**

```
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

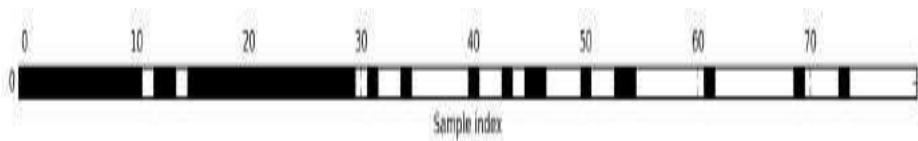
**Out[43]:**

```
X_train.shape: (284, 80)
X_train_l1.shape: (284, 40)
```

Sekali lagi, kita dapat melihat fitur yang dipilih (Gambar 4.10):

**In[44]:**

```
mask = select.get_support()
# visualize the mask -- black is True, white is False
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Sample index")
```



**Gambar 4.10** Fitur yang dipilih oleh SelectFromModel menggunakan RandomForestClassifier

Kali ini, semua kecuali dua fitur asli dipilih. Karena kami menetapkan untuk memilih 40 fitur, beberapa fitur noise juga dipilih. Mari kita lihat performanya:

**In[45]:**

```
X_test_l1 = select.transform(X_test)
score = LogisticRegression().fit(X_train_l1, y_train).score(X_test_l1, y_test)
print("Test score: {:.3f}".format(score))
```

**Out[45]:**

```
Test score: 0.951
```

Dengan pemilihan fitur yang lebih baik, kami juga memperoleh beberapa peningkatan di sini.

## 4.9 PEMILIHAN FITUR ITERATIF

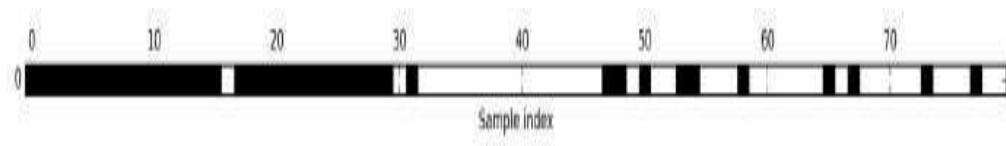
Dalam pengujian univariat kami tidak menggunakan model, sedangkan dalam pemilihan berbasis model kami menggunakan model tunggal untuk memilih fitur. Dalam pemilihan fitur berulang, serangkaian model dibangun, dengan jumlah fitur yang bervariasi. Ada dua metode dasar: memulai tanpa fitur dan menambahkan fitur satu per satu hingga beberapa kriteria penghentian tercapai, atau memulai dengan semua fitur dan menghapus fitur satu per satu hingga beberapa kriteria penghentian tercapai. Karena serangkaian model

dibangun, metode ini jauh lebih mahal secara komputasi daripada metode yang telah kita bahas sebelumnya. Salah satu metode khusus semacam ini adalah eliminasi fitur rekursif (RFE), yang dimulai dengan semua fitur, membangun model, dan membuang fitur yang paling tidak penting menurut model. Kemudian model baru dibangun menggunakan semua kecuali fitur yang dibuang, dan seterusnya sampai hanya sejumlah fitur yang telah ditentukan yang tersisa. Agar ini berfungsi, model yang digunakan untuk seleksi perlu menyediakan beberapa cara untuk menentukan pentingnya fitur, seperti halnya untuk pemilihan berbasis model. Di sini, kami menggunakan model hutan acak yang sama yang kami gunakan sebelumnya, dan mendapatkan hasil yang ditunjukkan pada Gambar 4.11:

**In[46]:**

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100, random_state=42),
              n_features_to_select=40)

select.fit(X_train, y_train)
# visualize the selected features:
mask = select.get_support()
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("Sample index")
```



**Gambar 4.11** Fitur yang dipilih dengan eliminasi fitur rekursif dengan model pengklasifikasi hutan acak

Pemilihan fitur menjadi lebih baik dibandingkan dengan pemilihan univariat dan berbasis model, tetapi satu fitur masih terlewatkan. Menjalankan kode ini juga membutuhkan waktu yang jauh lebih lama daripada pemilihan berbasis model, karena model hutan acak dilatih 40 kali, sekali untuk setiap fitur yang dijatuhan. Mari kita uji akurasi model regresi logistik saat menggunakan RFE untuk pemilihan fitur:

**In[47]:**

```
X_train_rfe= select.transform(X_train)
X_test_rfe= select.transform(X_test)

score = LogisticRegression().fit(X_train_rfe, y_train).score(X_test_rfe, y_test)
print("Test score: {:.3f}".format(score))
```

**Out[47]:**

```
Test score: 0.951
```

Kita juga dapat menggunakan model yang digunakan di dalam RFE untuk membuat prediksi. Ini hanya menggunakan set fitur yang dipilih:

In[48]:

```
print("Test score: {:.3f}".format(select.score(X_test, y_test)))
```

Out[48]:

```
Test score: 0.951
```

Di sini, kinerja hutan acak yang digunakan di dalam RFE sama dengan yang dicapai dengan melatih model regresi logistik di atas fitur yang dipilih. Dengan kata lain, setelah kami memilih fitur yang tepat, model linier berkinerja sebaik hutan acak.

Jika Anda tidak yakin saat memilih apa yang akan digunakan sebagai input ke algoritma pembelajaran mesin Anda, pemilihan fitur otomatis bisa sangat membantu. Ini juga bagus untuk mengurangi jumlah fitur yang diperlukan—misalnya, untuk mempercepat prediksi atau memungkinkan model yang lebih dapat diinterpretasikan. Dalam kebanyakan kasus dunia nyata, menerapkan pemilihan fitur tidak mungkin memberikan keuntungan besar dalam kinerja. Namun, itu masih merupakan alat yang berharga di kotak peralatan insinyur fitur.

#### 4.10 MEMANFAATKAN PENGETAHUAN AHLI

Rekayasa fitur sering menjadi tempat penting untuk menggunakan pengetahuan ahli untuk aplikasi tertentu. Sementara tujuan pembelajaran mesin dalam banyak kasus adalah untuk menghindari keharusan membuat seperangkat aturan yang dirancang ahli, itu tidak berarti bahwa pengetahuan sebelumnya tentang aplikasi atau domain harus dibuang. Seringkali, pakar domain dapat membantu mengidentifikasi fitur berguna yang jauh lebih informatif daripada representasi awal data. Bayangkan Anda bekerja untuk agen perjalanan dan ingin memprediksi harga penerbangan. Katakanlah Anda memiliki catatan harga bersama dengan tanggal, maskapai penerbangan, lokasi awal, dan tujuan. Model pembelajaran mesin mungkin dapat membangun model yang layak dari itu. Beberapa faktor penting dalam harga penerbangan, bagaimanapun, tidak dapat dipelajari. Misalnya, penerbangan biasanya lebih mahal selama bulan-bulan liburan puncak dan sekitar hari libur.

Sementara tanggal dari beberapa hari libur (seperti Natal) adalah tetap, dan karena itu pengaruhnya dapat dipelajari dari tanggal tersebut, yang lain mungkin bergantung pada fase bulan (seperti Hanukkah dan Paskah) atau ditetapkan oleh pihak berwenang (seperti hari libur sekolah). Peristiwa ini tidak dapat dipelajari dari data jika setiap penerbangan hanya direkam menggunakan tanggal (Gregorian). Namun, mudah untuk menambahkan fitur yang mengkodekan apakah penerbangan sedang berlangsung, sebelum, atau setelah hari libur umum atau sekolah. Dengan cara ini, pengetahuan sebelumnya tentang sifat tugas dapat dikodekan dalam fitur untuk membantu algoritma pembelajaran mesin. Menambahkan fitur tidak memaksa algoritme pembelajaran mesin untuk menggunakannya, dan bahkan jika informasi hari libur ternyata tidak informatif untuk harga penerbangan, menambahkan data dengan informasi ini tidak ada salahnya.

Sekarang kita akan melihat satu kasus tertentu dalam menggunakan pengetahuan ahli—meskipun dalam kasus ini mungkin lebih tepat disebut "akal sehat". Tugasnya adalah memprediksi persewaan sepeda di depan rumah Andreas.

Di New York, Citi Bike mengoperasikan jaringan stasiun penyewaan sepeda dengan sistem berlangganan. Stasiun-stasiunnya ada di seluruh kota dan menyediakan cara yang nyaman untuk berkeliling. Data persewaan sepeda dipublikasikan dalam bentuk anonim dan telah dianalisis dengan berbagai cara. Tugas yang ingin kami selesaikan adalah memprediksi pada waktu dan hari tertentu berapa banyak orang yang akan menyewa sepeda di depan rumah Andreas—jadi dia tahu jika ada sepeda yang akan ditinggalkan untuknya.

Kami pertama-tama memuat data untuk Agustus 2015 untuk stasiun khusus ini sebagai Bingkai Data panda. Kami mengambil sampel ulang data ke dalam interval tiga jam untuk mendapatkan tren utama setiap hari:

**In[49]:**

```
citibike = mglearn.datasets.load_citibike()
```

**In[50]:**

```
print("Citi Bike data:\n{}".format(citibike.head()))
```

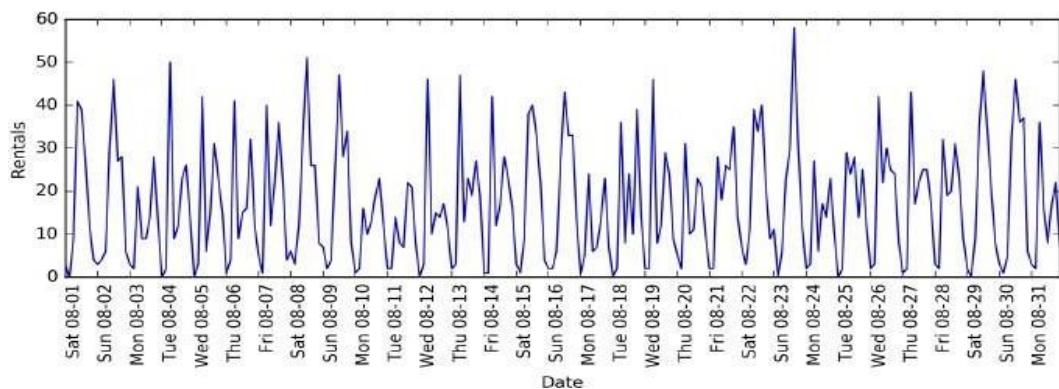
**Out[50]:**

```
Citi Bike data:
starttime
2015-08-01 00:00:00      3.0
2015-08-01 03:00:00      0.0
2015-08-01 06:00:00      9.0
2015-08-01 09:00:00     41.0
2015-08-01 12:00:00     39.0
Freq: 3H, Name: one, dtype: float64
```

Contoh berikut menunjukkan visualisasi frekuensi sewa selama sebulan penuh (Gambar 4.12):

**In[51]:**

```
plt.figure(figsize=(10, 3))
xticks = pd.date_range(start=citibike.index.min(), end=citibike.index.max(),
                       freq='D')
plt.xticks(xticks, xticks.strftime("%a %m-%d"), rotation=90, ha="left")
plt.plot(citibike, linewidth=1)
plt.xlabel("Date")
plt.ylabel("Rentals")
```



**Gambar 4.12** Jumlah rental sepeda dari waktu ke waktu untuk stasiun Citi Bike yang dipilih  
Machine Learning (Dr. Budi Raharjo)

Melihat data tersebut, kita dapat dengan jelas membedakan siang dan malam untuk setiap interval 24 jam. Pola untuk hari kerja dan akhir pekan juga tampaknya sangat berbeda. Saat mengevaluasi tugas prediksi pada rangkaian waktu seperti ini, biasanya kita ingin belajar dari masa lalu dan memprediksi masa depan. Ini berarti saat melakukan pemisahan menjadi set pelatihan dan pengujian, kami ingin menggunakan semua data hingga tanggal tertentu sebagai set pelatihan dan semua data yang melewati tanggal tersebut sebagai set pengujian. Beginilah cara kami biasanya menggunakan prediksi deret waktu: mengingat semua yang kami ketahui tentang persewaan di masa lalu, menurut kami apa yang akan terjadi besok? Kami akan menggunakan 184 titik data pertama, sesuai dengan 23 hari pertama, sebagai set pelatihan kami, dan 64 poin data tersisa, sesuai dengan 8 hari tersisa, sebagai set pengujian kami.

Satu-satunya fitur yang kami gunakan dalam tugas prediksi kami adalah tanggal dan waktu ketika sejumlah persewaan terjadi. Jadi, fitur inputnya adalah tanggal dan waktu—misalnya, 01-08-2015 00:00:00—and outputnya adalah jumlah rental dalam tiga jam berikutnya (tiga dalam kasus ini, menurut DataFrame kami).

Cara umum (yang mengejutkan) bahwa tanggal disimpan di komputer menggunakan waktu POSIX, yang merupakan jumlah detik sejak Januari 1970 00:00:00 (alias awal waktu Unix). Sebagai percobaan pertama, kita dapat menggunakan fitur integer tunggal ini sebagai representasi data kita:

**In[52]:**

```
# extract the target values (number of rentals)
y = citibike.values
# convert the time to POSIX time using "%s"
X = citibike.index.strftime("%s").astype("int").reshape(-1, 1)
```

Pertama-tama kita mendefinisikan fungsi untuk membagi data menjadi set pelatihan dan pengujian, membangun model, dan memvisualisasikan hasilnya:

**In[54]:**

```
# use the first 184 data points for training, and the rest for testing
n_train = 184

# function to evaluate and plot a regressor on a given feature set
def eval_on_features(features, target, regressor):
    # split the given features into a training and a test set
    X_train, X_test = features[:n_train], features[n_train:]
    # also split the target array
    y_train, y_test = target[:n_train], target[n_train:]
    regressor.fit(X_train, y_train)
    print("Test-set R^2: {:.2f}".format(regressor.score(X_test, y_test)))
    y_pred = regressor.predict(X_test)
    y_pred_train = regressor.predict(X_train)
    plt.figure(figsize=(10, 3))

    plt.xticks(range(0, len(X), 8), xticks.strftime("%a %m-%d"), rotation=90,
               ha="left")

    plt.plot(range(n_train), y_train, label="train")
    plt.plot(range(n_train, len(y_test) + n_train), y_test, '--', label="test")
    plt.plot(range(n_train), y_pred_train, '--', label="prediction train")
```

```

plt.plot(range(n_train, len(y_test) + n_train), y_pred, '--',
         label="prediction test")
plt.legend(loc=(1.01, 0))
plt.xlabel("Date")
plt.ylabel("Rentals")

```

Kita telah melihat sebelumnya bahwa hutan acak memerlukan sangat sedikit pra-pemrosesan data, yang membuat ini tampak seperti model yang baik untuk memulai. Kami menggunakan fitur waktu POSIX X dan meneruskan regressor hutan acak ke fungsi eval\_on\_features kami. Gambar 4.13 menunjukkan hasil:

**In[55]:**

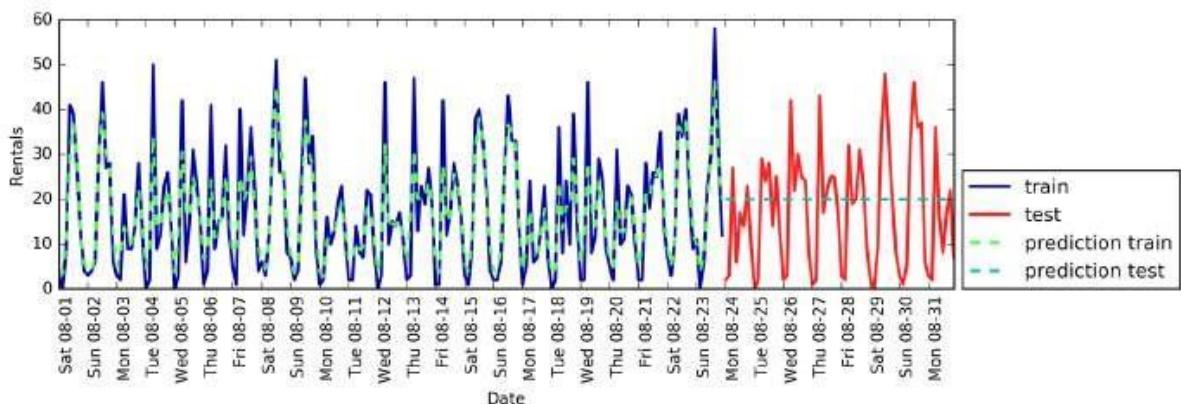
```

from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=0)
plt.figure()
eval_on_features(X, y, regressor)

```

**Out[55]:**

Test-set R<sup>2</sup>: -0.04



**Gambar 4.13** Prediksi yang dibuat oleh hutan acak hanya menggunakan waktu POSIX

Prediksi pada set pelatihan cukup bagus, seperti biasa untuk hutan acak. Namun, untuk set tes, garis konstan diprediksi. R<sup>2</sup> adalah -0,03, yang berarti kita tidak belajar apa-apa. Apa yang terjadi?

Masalahnya terletak pada kombinasi fitur kami dan hutan acak. Nilai fitur waktu POSIX untuk set pengujian berada di luar rentang nilai fitur dalam set pelatihan: poin dalam set pengujian memiliki stempel waktu yang lebih lambat dari semua poin dalam set pelatihan. Pohon, dan karena itu hutan acak, tidak dapat diekstrapolasi ke rentang fitur di luar set pelatihan. Hasilnya adalah model hanya memprediksi nilai target dari titik terdekat dalam set pelatihan—yang merupakan terakhir kalinya ia mengamati data apa pun.

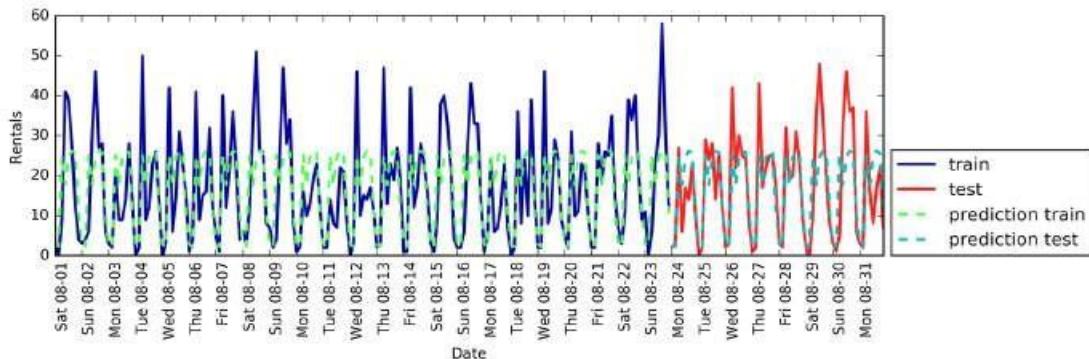
Jelas kami bisa lebih baik dari ini. Di sinilah "pengetahuan ahli" kami masuk. Dari melihat angka sewa dalam data pelatihan, dua faktor tampaknya sangat penting: waktu dan hari dalam seminggu. Jadi, mari tambahkan dua fitur ini. Kami tidak dapat benar-benar mempelajari apa pun dari waktu POSIX, jadi kami menghapus fitur itu. Pertama, mari kita gunakan hanya jam dalam sehari. Seperti yang ditunjukkan Gambar 4.14, sekarang prediksi memiliki pola yang sama untuk setiap hari dalam seminggu:

**In[56]:**

```
X_hour = citibike.index.hour.reshape(-1, 1)
eval_on_features(X_hour, y, regressor)
```

**Out[56]:**

Test-set R<sup>2</sup>: 0.60



**Gambar 4.14** Prediksi yang dibuat oleh hutan acak hanya menggunakan jam dalam sehari

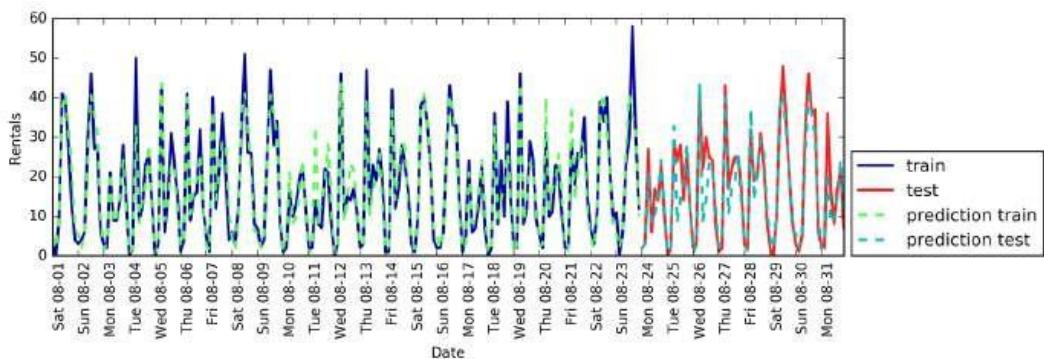
R<sup>2</sup> sudah jauh lebih baik, tetapi prediksinya jelas meleset dari pola mingguan. Sekarang mari kita tambahkan juga hari dalam seminggu (lihat Gambar 4.15):

**In[57]:**

```
X_hour_week = np.hstack([citibike.index.dayofweek.reshape(-1, 1),
                         citibike.index.hour.reshape(-1, 1)])
eval_on_features(X_hour_week, y, regressor)
```

**Out[57]:**

Test-set R<sup>2</sup>: 0.84



**Gambar 4.15** Prediksi dengan hutan acak menggunakan fitur hari dalam seminggu dan jam

Sekarang kita memiliki model yang menangkap perilaku periodik dengan mempertimbangkan hari dalam seminggu dan waktu dalam sehari. Ini memiliki R<sup>2</sup> 0,84, dan menunjukkan kinerja prediksi yang cukup baik. Apa yang kemungkinan dipelajari oleh model ini adalah jumlah rata-rata persewaan untuk setiap kombinasi hari kerja dan waktu dari 23 hari pertama bulan Agustus. Ini sebenarnya tidak memerlukan model yang kompleks seperti

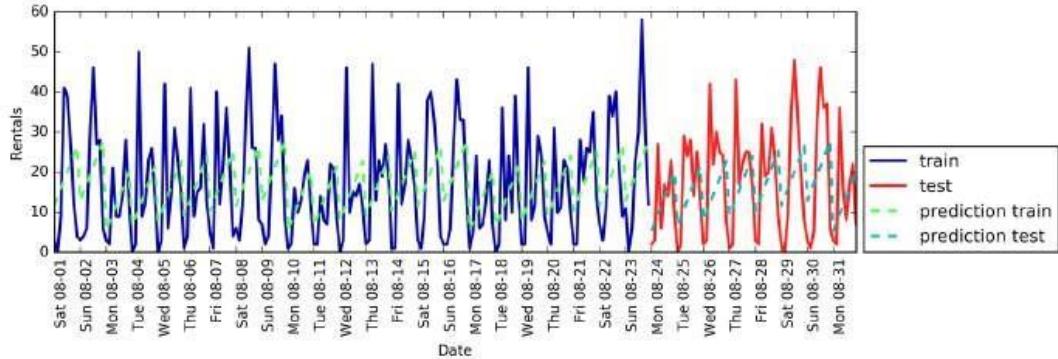
hutan acak, jadi mari kita coba dengan model yang lebih sederhana, LinearRegression (lihat Gambar 4.16):

**In[58]:**

```
from sklearn.linear_model import LinearRegression
eval_on_features(X_hour_week, y, LinearRegression())
```

**Out[58]:**

```
Test-set R^2: 0.13
```



**Gambar 4.16** Prediksi yang dibuat dengan regresi linier menggunakan hari dalam seminggu dan jam sebagai fitur

LinearRegression bekerja jauh lebih buruk, dan pola periodik terlihat aneh. Alasan untuk ini adalah bahwa kami mengkodekan hari dalam seminggu dan waktu menggunakan bilangan bulat, yang ditafsirkan sebagai variabel kategoris. Oleh karena itu, model linier hanya dapat mempelajari fungsi linier dari waktu—and diketahui bahwa di kemudian hari, ada lebih banyak persewaan. Namun, polanya jauh lebih kompleks dari itu. Kita dapat menangkap ini dengan menafsirkan bilangan bulat sebagai variabel kategori, dengan mentransformasikannya menggunakan OneHotEncoder (lihat Gambar 4.17):

**In[59]:**

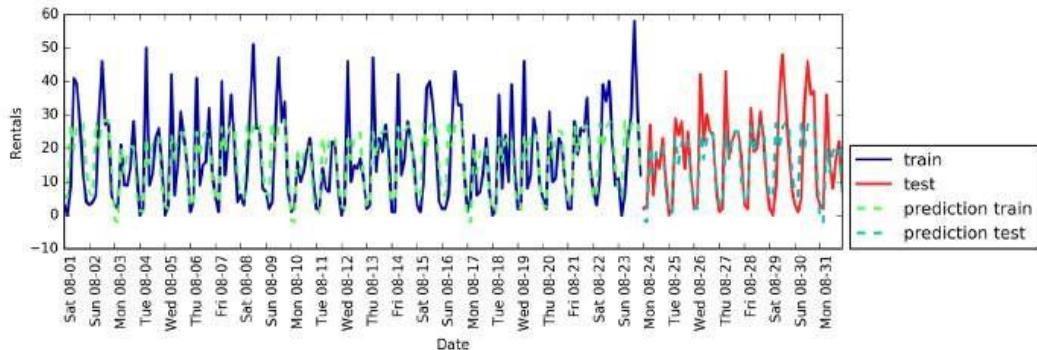
```
enc = OneHotEncoder()
X_hour_week_onehot = enc.fit_transform(X_hour_week).toarray()
```

**In[60]:**

```
eval_on_features(X_hour_week_onehot, y, Ridge())
```

**Out[60]:**

```
Test-set R^2: 0.62
```



**Gambar 4.17** Prediksi yang dibuat dengan regresi linier menggunakan pengkodean satu-panas dari jam hari dan hari dalam seminggu

Ini memberi kami kecocokan yang jauh lebih baik daripada pengkodean fitur berkelanjutan. Sekarang model linier mempelajari satu koefisien untuk setiap hari dalam seminggu, dan satu koefisien untuk setiap waktu dalam sehari. Itu berarti bahwa pola "waktu dalam sehari" dibagikan sepanjang hari dalam seminggu.

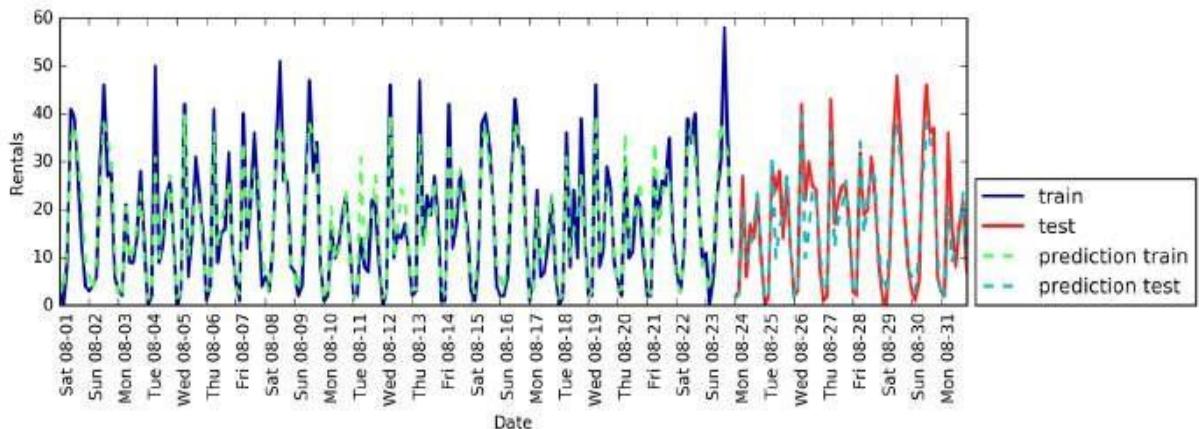
Dengan menggunakan fitur interaksi, kita dapat mengizinkan model untuk mempelajari satu koefisien untuk setiap kombinasi hari dan waktu (lihat Gambar 4.18):

**In[61]:**

```
poly_transformer = PolynomialFeatures(degree=2, interaction_only=True,
                                     include_bias=False)
X_hour_week_onehot_poly = poly_transformer.fit_transform(X_hour_week_onehot)
lr = Ridge()
eval_on_features(X_hour_week_onehot_poly, y, lr)
```

**Out[61]:**

```
Test-set R^2: 0.85
```



**Gambar 4.18** Prediksi yang dibuat dengan regresi linier menggunakan fitur produk hari dalam seminggu dan jam

Transformasi ini akhirnya menghasilkan model yang berkinerja baik dengan hutan acak. Manfaat besar dari model ini adalah sangat jelas apa yang dipelajari: satu koefisien untuk setiap hari dan waktu. Kita dapat dengan mudah memplot koefisien yang dipelajari oleh

model, sesuatu yang tidak mungkin untuk hutan acak. Pertama, kami membuat nama fitur untuk fitur jam dan hari:

**In[62]:**

```
hour = ["%02d:00" % i for i in range(0, 24, 3)]
day = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
features = day + hour
```

Kemudian kami memberi nama semua fitur interaksi yang diekstraksi oleh PolynomialFeatures, menggunakan metode get\_feature\_names, dan hanya menyimpan fitur dengan koefisien bukan nol:

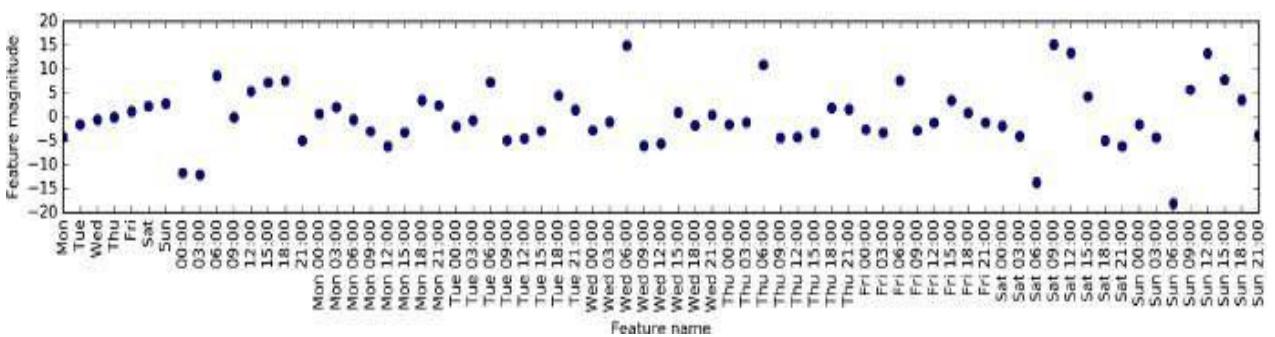
**In[63]:**

```
features_poly = poly_transformer.get_feature_names(features)
features_nonzero = np.array(features_poly)[lr.coef_ != 0]
coef_nonzero = lr.coef_[lr.coef_ != 0]
```

Sekarang kita dapat memvisualisasikan koefisien yang dipelajari oleh model linier, seperti yang terlihat pada Gambar 4.19:

**In[64]:**

```
plt.figure(figsize=(15, 2))
plt.plot(coef_nonzero, 'o')
plt.xticks(np.arange(len(coef_nonzero)), features_nonzero, rotation=90)
plt.xlabel("Feature magnitude")
plt.ylabel("Feature")
```



**Gambar 4.19** Koefisien model regresi linier menggunakan produk jam dan hari

#### 4.11 RINGKASAN DAN PANDANGAN

Dalam bab ini, kita membahas bagaimana menangani tipe data yang berbeda (khususnya, dengan variabel kategori). Kami menekankan pentingnya merepresentasikan data dengan cara yang sesuai untuk algoritme pembelajaran mesin—misalnya, dengan variabel kategorikal enkode satu-panas. Kami juga membahas pentingnya merekayasa fitur baru, dan kemungkinan memanfaatkan pengetahuan ahli dalam membuat fitur turunan dari data Anda. Secara khusus, model linier mungkin sangat diuntungkan dari menghasilkan fitur baru melalui binning dan menambahkan polinomial dan interaksi, sementara model nonlinier yang lebih kompleks seperti hutan acak dan SVM mungkin dapat mempelajari tugas yang lebih kompleks tanpa secara eksplisit memperluas ruang fitur. Dalam praktiknya, fitur yang

digunakan (dan kecocokan antara fitur dan metode) seringkali merupakan bagian terpenting dalam membuat pendekatan pembelajaran mesin bekerja dengan baik.

Sekarang setelah Anda memiliki ide yang bagus tentang cara merepresentasikan data Anda dengan cara yang tepat dan algoritma mana yang digunakan untuk tugas mana, bab berikutnya akan fokus pada evaluasi kinerja model pembelajaran mesin dan memilih pengaturan parameter yang tepat.

## BAB 5

### EVALUASI DAN PENINGKATAN MODEL

Setelah membahas dasar-dasar pembelajaran terawasi dan tidak terawasi, dan setelah menjelajahi berbagai algoritma pembelajaran mesin, sekarang kita akan menyelam lebih dalam ke dalam mengevaluasi model dan memilih parameter.

Kami akan fokus pada metode yang diawasi, regresi dan klasifikasi, karena mengevaluasi dan memilih model dalam pembelajaran tanpa pengawasan seringkali merupakan proses yang sangat kualitatif (seperti yang kita lihat di Bab 3).

Untuk mengevaluasi model terawasi kami, sejauh ini kami telah membagi set data menjadi set pelatihan dan set pengujian menggunakan fungsi `train_test_split`, membangun model pada set pelatihan dengan memanggil metode `fit`, dan mengevaluasinya pada set pengujian menggunakan metode skor, yang untuk klasifikasi menghitung fraksi sampel yang diklasifikasikan dengan benar. Berikut adalah contoh dari proses itu:

**In[2]:**

```
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# create a synthetic dataset
X, y = make_blobs(random_state=0)
# split data and labels into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)
# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))
```

**Out[2]:**

```
Test set score: 0.88
```

Ingat, alasan kami membagi data menjadi set pelatihan dan pengujian adalah karena kami tertarik untuk mengukur seberapa baik model kami digeneralisasikan ke data baru yang sebelumnya tidak terlihat. Kami tidak tertarik pada seberapa baik model kami cocok dengan set pelatihan, tetapi lebih pada seberapa baik model dapat membuat prediksi untuk data yang tidak diamati selama pelatihan.

Dalam bab ini, kami akan memperluas dua aspek evaluasi ini. Kami pertama-tama akan memperkenalkan validasi silang, cara yang lebih kuat untuk menilai kinerja generalisasi, dan mendiskusikan metode untuk mengevaluasi kinerja klasifikasi dan regresi yang melampaui ukuran standar akurasi dan R<sup>2</sup> yang disediakan oleh metode skor.

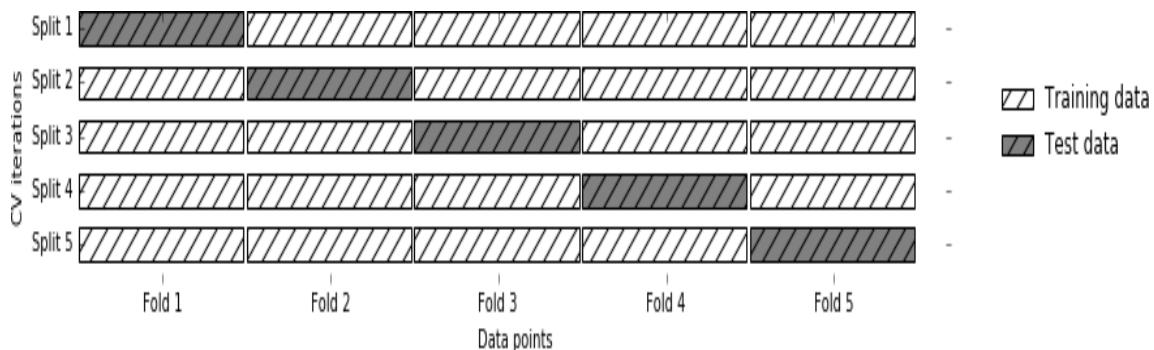
Kami juga akan membahas pencarian grid, metode yang efektif untuk menyesuaikan parameter dalam model yang diawasi untuk kinerja generalisasi terbaik.

## 5.1 VALIDASI SILANG

Validasi silang adalah metode statistik untuk mengevaluasi kinerja generalisasi yang lebih stabil dan menyeluruh daripada menggunakan pemisahan menjadi set pelatihan dan pengujian. Dalam validasi silang, data dipecah berulang kali dan beberapa model dilatih. Versi validasi silang yang paling umum digunakan adalah validasi silang k-fold, di mana k adalah nomor yang ditentukan pengguna, biasanya 5 atau 10. Saat melakukan validasi silang lima kali, data pertama-tama dipartisi menjadi lima bagian dari (kira-kira) ukuran yang sama, yang disebut lipatan. Selanjutnya, urutan model dilatih. Model pertama dilatih menggunakan lipatan pertama sebagai set tes, dan lipatan yang tersisa (2–5) digunakan sebagai set pelatihan. Model dibangun menggunakan data pada lipatan 2–5, kemudian akurasinya dievaluasi pada lipatan 1. Kemudian model lain dibangun, kali ini menggunakan set uji lipatan 2 dan data pada lipatan 1, 3, 4, dan 5 sebagai set pelatihan. Proses ini diulang menggunakan lipatan 3, 4, dan 5 sebagai set tes. Untuk masing-masing dari lima pemisahan data ini menjadi set pelatihan dan pengujian, kami menghitung akurasinya. Pada akhirnya, kami telah mengumpulkan lima nilai akurasi. Prosesnya diilustrasikan pada Gambar 5.1:

In[3]:

```
mglearn.plots.plot_cross_validation()
```



Gambar 5.1 Pemisahan data dalam validasi silang lima kali lipat

Biasanya, seperlima pertama dari data adalah lipatan pertama, seperlima kedua dari data adalah lipatan kedua, dan seterusnya.

### Validasi Silang dalam scikit-learn

Validasi silang diimplementasikan dalam scikit-learn menggunakan fungsi `cross_val_score` dari modul `model_selection`. Parameter fungsi `cross_val_score` adalah model yang ingin kita evaluasi, data pelatihan, dan label kebenaran dasar. Mari kita evaluasi `LogisticRegression` pada dataset iris:

In[4]:

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression()
```

```
scores = cross_val_score(logreg, iris.data, iris.target)
print("Cross-validation scores: {}".format(scores))
```

**Out[4]:**

```
Cross-validation scores: [ 0.961  0.922  0.958]
```

Secara default, `cross_val_score` melakukan validasi silang tiga kali lipat, mengembalikan tiga nilai akurasi. Kita dapat mengubah jumlah lipatan yang digunakan dengan mengubah parameter `cv`:

**In[5]:**

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("Cross-validation scores: {}".format(scores))
```

**Out[5]:**

```
Cross-validation scores: [ 1.      0.967  0.933  0.9     1.     ]
```

Cara umum untuk meringkas akurasi validasi silang adalah dengan menghitung mean:

**In[6]:**

```
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

**Out[6]:**

```
Average cross-validation score: 0.96
```

Dengan menggunakan validasi silang rata-rata, kami dapat menyimpulkan bahwa kami mengharapkan model rata-rata sekitar 96% akurat. Melihat kelima skor yang dihasilkan oleh validasi silang lima kali lipat, kami juga dapat menyimpulkan bahwa ada perbedaan yang relatif tinggi dalam akurasi antar lipatan, mulai dari akurasi 100% hingga akurasi 90%. Ini bisa berarti bahwa model sangat bergantung pada lipatan tertentu yang digunakan untuk pelatihan, tetapi bisa juga karena ukuran kecil dari kumpulan data.

### Manfaat Cross-Validasi

Ada beberapa manfaat menggunakan validasi silang daripada pemisahan tunggal menjadi set pelatihan dan pengujian. Pertama, ingat bahwa `train_test_split` melakukan pemisahan data secara acak. Bayangkan bahwa kita "beruntung" ketika memisahkan data secara acak, dan semua contoh yang sulit untuk diklasifikasikan berakhir di set pelatihan. Dalam hal ini, set pengujian hanya akan berisi contoh "mudah", dan akurasi set pengujian kami akan sangat tinggi. Sebaliknya, jika kita "tidak beruntung", kita mungkin secara acak memasukkan semua contoh yang sulit untuk diklasifikasikan ke dalam perangkat tes dan akibatnya memperoleh skor rendah yang tidak realistik. Namun, saat menggunakan validasi silang, setiap contoh akan berada di set pelatihan tepat satu kali: setiap contoh berada di salah satu lipatan, dan setiap lipatan adalah set pengujian satu kali. Oleh karena itu, model perlu digeneralisasi dengan baik ke semua sampel dalam kumpulan data agar semua skor validasi silang (dan rata-ratanya) menjadi tinggi.

Memiliki beberapa pemisahan data juga memberikan beberapa informasi tentang seberapa sensitif model kami terhadap pemilihan set data pelatihan. Untuk dataset iris, kami melihat akurasi antara 90% dan 100%. Ini adalah rentang yang cukup besar, dan ini memberi

kita gambaran tentang bagaimana model dapat bekerja dalam skenario terburuk dan skenario terbaik ketika diterapkan pada data baru.

Manfaat lain dari validasi silang dibandingkan dengan menggunakan satu pemisahan data adalah bahwa kami menggunakan data kami secara lebih efektif. Saat menggunakan `train_test_split`, kami biasanya menggunakan 75% data untuk pelatihan dan 25% data untuk evaluasi. Saat menggunakan validasi silang lima kali lipat, dalam setiap iterasi kita dapat menggunakan empat perlama dari data (80%) agar sesuai dengan model. Saat menggunakan validasi silang 10 kali lipat, kita dapat menggunakan sembilan per sepuluh data (90%) agar sesuai dengan model. Lebih banyak data biasanya akan menghasilkan model yang lebih akurat.

Kerugian utama dari validasi silang adalah peningkatan biaya komputasi. Karena kita sekarang melatih k model dan bukan satu model, validasi silang akan kira-kira k kali lebih lambat daripada melakukan satu pemisahan data.



Penting untuk diingat bahwa validasi silang bukanlah cara untuk membangun model yang dapat diterapkan pada data baru. Validasi silang tidak mengembalikan model. Saat memanggil `cross_val_score`, beberapa model dibangun secara internal, tetapi tujuan validasi silang hanya untuk mengevaluasi seberapa baik algoritme yang diberikan akan digeneralisasi ketika dilatih pada kumpulan data tertentu.

## 5.2 STRATIFIED K-FOLD CROSS-VALIDATION DAN STRATEGI LAINNYA

Memisahkan kumpulan data menjadi k lipatan dengan memulai dengan bagian sepe-  
k pertama dari data, seperti yang dijelaskan di bagian sebelumnya, mungkin tidak selalu  
merupakan ide yang baik. Sebagai contoh, mari kita lihat dataset iris:

In[7]:

```
from sklearn.datasets import load_iris  
iris = load_iris()  
print("Iris labels:\n{}".format(iris.target))
```

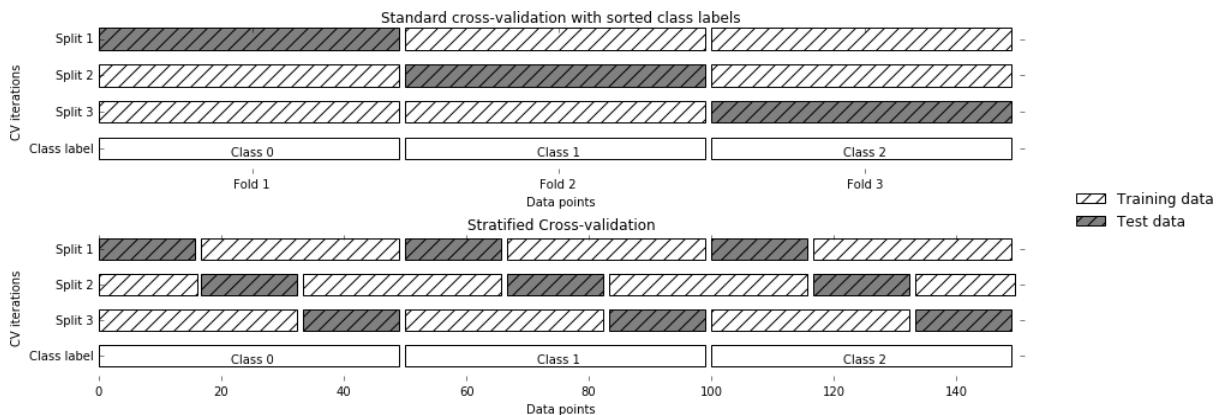
Out[7]:

Seperti yang Anda lihat, sepertiga pertama dari data adalah kelas 0, sepertiga kedua adalah kelas 1, dan sepertiga terakhir adalah kelas 2. Bayangkan melakukan validasi silang tiga kali lipat pada dataset ini. Lipatan pertama hanya akan menjadi kelas 0, jadi pada pemisahan data pertama, set tes hanya akan menjadi kelas 0, dan set pelatihan hanya akan menjadi kelas 1 dan 2. Karena kelas dalam set pelatihan dan tes akan berbeda untuk ketiga pemisahan, akurasi validasi silang tiga kali lipat akan menjadi nol pada kumpulan data ini. Itu tidak terlalu membantu, karena kami dapat melakukan jauh lebih baik daripada akurasi 0% pada iris.

Karena strategi k-fold sederhana gagal di sini, scikit-learn tidak menggunakannya untuk klasifikasi, melainkan menggunakan validasi silang k-fold bertingkat. Dalam validasi silang bertingkat, kami membagi data sedemikian rupa sehingga proporsi antar kelas sama di setiap lipatan seperti halnya di seluruh kumpulan data, seperti yang diilustrasikan pada Gambar 5.2:

**In[8]:**

```
mglearn.plots.plot_stratified_cross_validation()
```



**Gambar 5.2** Perbandingan validasi silang standar dan validasi silang bertingkat ketika data diurutkan berdasarkan label kelas

Misalnya, jika 90% sampel Anda termasuk dalam kelas A dan 10% sampel Anda termasuk dalam kelas B, maka validasi silang bertingkat memastikan bahwa di setiap lipatan, 90% sampel termasuk dalam kelas A dan 10% sampel milik kelas B.

Biasanya merupakan ide yang baik untuk menggunakan validasi silang k-fold bertingkat daripada validasi silang k-fold untuk mengevaluasi pengklasifikasi, karena ini menghasilkan perkiraan kinerja generalisasi yang lebih andal. Dalam kasus hanya 10% sampel yang termasuk dalam kelas B, menggunakan validasi silang k-fold standar, mungkin dengan mudah terjadi bahwa satu lipatan hanya berisi sampel kelas A. Menggunakan lipatan ini sebagai set tes tidak akan terlalu informatif tentang kinerja keseluruhan classifier.

Untuk regresi, scikit-learn menggunakan validasi silang k-fold standar secara default. Mungkin juga untuk mencoba membuat setiap lipatan mewakili nilai berbeda yang dimiliki target regresi, tetapi ini bukan strategi yang umum digunakan dan akan mengejutkan bagi sebagian besar pengguna.

### Lebih banyak kontrol atas validasi silang

Kita telah melihat sebelumnya bahwa kita dapat menyesuaikan jumlah lipatan yang digunakan dalam `cross_val_score` menggunakan parameter `cv`. Namun, scikit-learn memungkinkan kontrol yang lebih baik atas apa yang terjadi selama pemisahan data dengan menyediakan pembagi validasi silang sebagai parameter `cv`. Untuk sebagian besar kasus penggunaan, default validasi silang k-fold untuk regresi dan k-fold bertingkat untuk klasifikasi berfungsi dengan baik, tetapi ada beberapa kasus di mana mungkin ingin menggunakan strategi yang berbeda. Katakanlah, misalnya, kami ingin menggunakan validasi silang k-fold standar pada dataset klasifikasi untuk mereproduksi hasil orang lain. Untuk melakukan ini,

pertama-tama kita harus mengimpor kelas splitter KFold dari modul model\_selection dan membuat instance dengan jumlah lipatan yang ingin kita gunakan:

**In[9]:**

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=5)
```

Kemudian, kita dapat meneruskan objek kfold splitter sebagai parameter cv ke cross\_val\_score:

**In[10]:**

```
print("Cross-validation scores:\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

**Out[10]:**

```
Cross-validation scores:
[ 1.      0.933  0.433  0.967  0.433]
```

Dengan cara ini, kami dapat memverifikasi bahwa itu memang ide yang sangat buruk untuk menggunakan validasi silang tiga kali lipat (tidak bertingkat) pada set data iris:

**In[11]:**

```
kfold = KFold(n_splits=3)
print("Cross-validation scores:\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

**Out[11]:**

```
Cross-validation scores:
[ 0.  0.  0.]
```

Ingat: setiap lipatan sesuai dengan salah satu kelas dalam dataset iris, jadi tidak ada yang bisa dipelajari. Cara lain untuk mengatasi masalah ini adalah dengan mengacak data alih-alih membuat stratifikasi lipatan, untuk menghapus urutan sampel berdasarkan label. Kita dapat melakukannya dengan mengatur parameter shuffle KFold ke True. Jika kita mengacak data, kita juga perlu memperbaiki random\_state untuk mendapatkan pengacakan yang dapat direproduksi. Jika tidak, setiap menjalankan cross\_val\_score akan menghasilkan hasil yang berbeda, karena setiap kali pemisahan yang berbeda akan digunakan (ini mungkin tidak menjadi masalah, tetapi dapat mengejutkan). Mengacak data sebelum memisahkannya menghasilkan hasil yang jauh lebih baik:

**In[12]:**

```
kfold = KFold(n_splits=3, shuffle=True, random_state=0)
print("Cross-validation scores:\n{}".format(
    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))
```

**Out[12]:**

```
Cross-validation scores:
[ 0.9  0.96  0.96]
```

### Validasi silang *leave-one-out*

Metode validasi silang lain yang sering digunakan adalah leave-one-out. Anda dapat menganggap validasi silang *leave-one-out* sebagai validasi silang k-fold di mana setiap fold adalah satu sampel. Untuk setiap pemisahan, Anda memilih satu titik data untuk menjadi kumpulan pengujian. Ini bisa sangat memakan waktu, terutama untuk kumpulan data besar, tetapi terkadang memberikan perkiraan yang lebih baik untuk kumpulan data kecil:

**In[13]:**

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
print("Number of cv iterations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

**Out[13]:**

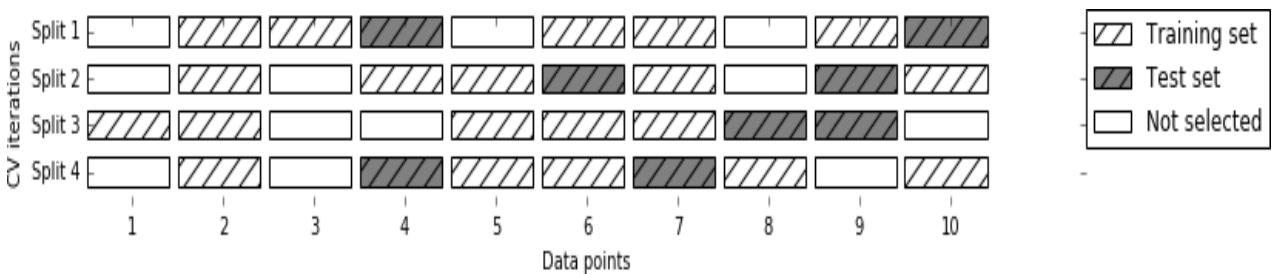
```
Number of cv iterations:  150
Mean accuracy: 0.95
```

### Validasi silang acak-split

Strategi lain yang sangat fleksibel untuk validasi silang adalah validasi silang shuffle-split. Dalam validasi silang shuffle-split, setiap sampel split melatih\_ukuran banyak poin untuk set pelatihan dan menguji\_ukuran banyak (berpisah) poin untuk set pengujian. Pemisahan ini diulang n\_iter kali. Gambar 5.3 mengilustrasikan menjalankan empat iterasi pemisahan set data yang terdiri dari 10 poin, dengan set pelatihan 5 poin dan set pengujian masing-masing 2 poin (Anda dapat menggunakan bilangan bulat untuk train\_size dan test\_size untuk menggunakan ukuran absolut untuk set ini, atau mengambah -poin angka untuk menggunakan pecahan dari seluruh dataset):

**In[14]:**

```
mglearn.plots.plot_shuffle_split()
```



**Gambar 5.3** ShuffleSplit dengan 10 poin, train\_size=5, test\_size=2, dan n\_iter=4

Kode berikut membagi dataset menjadi 50% training set dan 50% test set untuk 10 iterasi:

**In[15]:**

```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
print("Cross-validation scores:\n{}".format(scores))
```

**Out[15]:**

```
Cross-validation scores:
[ 0.96  0.907  0.947  0.96   0.96   0.907  0.893  0.907  0.92   0.973]
```

Validasi silang shuffle-split memungkinkan untuk mengontrol jumlah iterasi secara independen dari ukuran pelatihan dan pengujian, yang terkadang dapat membantu. Ini juga memungkinkan untuk menggunakan hanya sebagian dari data di setiap iterasi, dengan menyediakan pengaturan `train_size` dan `test_size` yang tidak bertambah satu. Subsampling data dengan cara ini dapat berguna untuk bereksperimen dengan kumpulan data yang besar.

Ada juga varian bertingkat dari `ShuffleSplit`, yang diberi nama `StratifiedShuffleSplit`, yang dapat memberikan hasil yang lebih andal untuk tugas klasifikasi.

### Validasi silang dengan grup

Pengaturan lain yang sangat umum untuk validasi silang adalah ketika ada kelompok dalam data yang sangat terkait. Katakanlah Anda ingin membangun sistem untuk mengenali emosi dari gambar wajah, dan Anda mengumpulkan kumpulan data gambar 100 orang di mana setiap orang ditangkap beberapa kali, menunjukkan berbagai emosi. Tujuannya adalah untuk membangun pengklasifikasi yang dapat mengidentifikasi emosi orang yang tidak ada dalam kumpulan data dengan benar. Anda dapat menggunakan validasi silang bertingkat default untuk mengukur kinerja pengklasifikasi di sini. Namun, kemungkinan gambar orang yang sama akan ada di set pelatihan dan tes. Akan jauh lebih mudah bagi pengklasifikasi untuk mendeteksi emosi di wajah yang merupakan bagian dari rangkaian pelatihan, dibandingkan dengan wajah yang sama sekali baru. Untuk secara akurat mengevaluasi generalisasi ke wajah baru, karena itu kita harus memastikan bahwa set pelatihan dan tes berisi gambar orang yang berbeda.

Untuk mencapai ini, kita dapat menggunakan `GroupKFold`, yang mengambil array grup sebagai argumen yang dapat kita gunakan untuk menunjukkan orang mana yang ada di dalam gambar. Array grup di sini menunjukkan grup dalam data yang tidak boleh dipisah saat membuat set pelatihan dan pengujian, dan tidak boleh dikacaukan dengan label kelas.

Contoh grup dalam data ini umum dalam aplikasi medis, di mana Anda mungkin memiliki beberapa sampel dari pasien yang sama, tetapi tertarik untuk menggeneralisasi ke pasien baru. Demikian pula, dalam pengenalan ucapan, Anda mungkin memiliki beberapa rekaman dari pembicara yang sama dalam kumpulan data Anda, tetapi tertarik untuk mengenali ucapan dari pembicara baru.

Berikut ini adalah contoh penggunaan dataset sintetis dengan pengelompokan yang diberikan oleh array `groups`. Kumpulan data terdiri dari 12 titik data, dan untuk setiap titik data, kelompok menentukan kelompok mana (berpikir sabar) titik tersebut. Kelompok menentukan bahwa ada empat kelompok, dan tiga sampel pertama termasuk dalam kelompok pertama, empat sampel berikutnya termasuk dalam kelompok kedua, dan seterusnya:

**In[17]:**

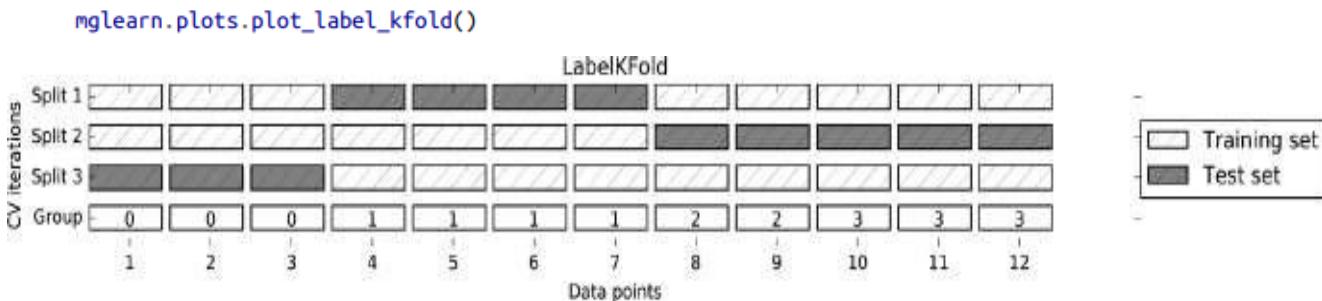
```
from sklearn.model_selection import GroupKFold
# create synthetic dataset
X, y = make_blobs(n_samples=12, random_state=0)
# assume the first three samples belong to the same group,
# then the next four, etc.
groups = [0, 0, 0, 1, 1, 1, 2, 2, 3, 3, 3]
scores = cross_val_score(logreg, X, y, groups, cv=GroupKFold(n_splits=3))
print("Cross-validation scores:\n{}".format(scores))
```

**Out[17]:**

```
Cross-validation scores:
[ 0.75  0.8   0.667]
```

Sampel tidak perlu dipesan berdasarkan kelompok; kami hanya melakukan ini untuk tujuan ilustrasi. Perpecahan yang dihitung berdasarkan label ini divisualisasikan pada Gambar 5.4. Seperti yang Anda lihat, untuk setiap pemisahan, setiap grup seluruhnya berada dalam set pelatihan atau seluruhnya dalam set pengujian:

**In[16]:**



**Gambar 5.4** Pemisahan yang bergantung pada label dengan GroupKFold

Ada lebih banyak strategi pemisahan untuk validasi silang di scikit-learn, yang memungkinkan lebih banyak variasi kasus penggunaan (Anda dapat menemukannya di panduan pengguna scikit-learn). Namun, KFold standar, StratifiedKFold, dan GroupKFold sejauh ini adalah yang paling umum digunakan.

### 5.3 PENCARIAN KOTAK

Sekarang kita tahu bagaimana mengevaluasi seberapa baik sebuah model menggeneralisasi, kita dapat mengambil langkah berikutnya dan meningkatkan kinerja generalisasi model dengan menyetel parameternya. Kami membahas pengaturan parameter dari banyak algoritme di scikit-learn di Bab 2 dan 3, dan penting untuk memahami apa arti parameter sebelum mencoba menyesuaikannya. Menemukan nilai parameter penting dari suatu model (yang memberikan kinerja generalisasi terbaik) adalah tugas yang rumit, tetapi diperlukan untuk hampir semua model dan kumpulan data. Karena ini adalah tugas yang umum, ada metode standar di scikit-learn untuk membantu Anda melakukannya. Metode yang paling umum digunakan adalah pencarian grid, yang pada dasarnya berarti mencoba semua kemungkinan kombinasi parameter yang diinginkan.

Pertimbangkan kasus kernel SVM dengan kernel RBF (fungsi dasar radial), seperti yang diterapkan di kelas SVC. Seperti yang telah kita bahas di Bab 2, ada dua parameter penting: bandwidth kernel, gamma, dan parameter regularisasi, C. Katakanlah kita ingin mencoba nilai 0,001, 0,01, 0,1, 1, 10, dan 100 untuk parameter C, dan hal yang sama untuk gamma. Karena kami memiliki enam pengaturan berbeda untuk C dan gamma yang ingin kami coba, kami memiliki total 36 kombinasi parameter. Melihat semua kemungkinan kombinasi, buat tabel (atau kisi) pengaturan parameter untuk SVM, seperti yang ditunjukkan di sini:

	$C = 0.001$	$C = 0.01$	$\dots$	$C = 10$
$\text{gamma}=0.001$	SVC( $C=0.001$ , $\text{gamma}=0.001$ )	SVC( $C=0.01$ , $\text{gamma}=0.001$ )	$\dots$	SVC( $C=10$ , $\text{gamma}=0.001$ )
$\text{gamma}=0.01$	SVC( $C=0.001$ , $\text{gamma}=0.01$ )	SVC( $C=0.01$ , $\text{gamma}=0.01$ )	$\dots$	SVC( $C=10$ , $\text{gamma}=0.01$ )
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$\text{gamma}=100$	SVC( $C=0.001$ , $\text{gamma}=100$ )	SVC( $C=0.01$ , $\text{gamma}=100$ )	$\dots$	SVC( $C=10$ , $\text{gamma}=100$ )

### Pencarian Kotak Sederhana

Kita dapat menerapkan pencarian grid sederhana seperti untuk loop pada dua parameter, melatih dan mengevaluasi classifier untuk setiap kombinasi:

In[18]:

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {} size of test set: {}".format(
    X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

Out[18]:

```
Size of training set: 112 size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

### 5.4 BAHAYA MELEBIHI PAMETER DAN SET VALIDASI

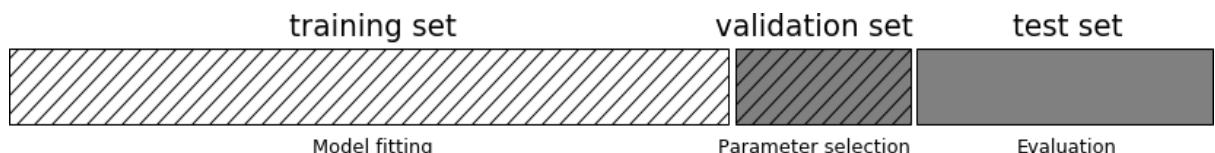
Mengingat hasil ini, kami mungkin tergoda untuk melaporkan bahwa kami menemukan model yang berkinerja dengan akurasi 97% pada kumpulan data kami. Namun, klaim ini bisa jadi terlalu optimis (atau hanya salah), karena alasan berikut: kami mencoba banyak parameter berbeda dan memilih salah satu dengan akurasi terbaik pada set pengujian,

tetapi akurasi ini tidak serta merta terbawa ke data baru. Karena kami menggunakan data uji untuk menyesuaikan parameter, kami tidak dapat lagi menggunakan untuk menilai seberapa bagus modelnya. Ini adalah alasan yang sama kami perlu membagi data menjadi set pelatihan dan pengujian; kita membutuhkan kumpulan data independen untuk dievaluasi, yang tidak digunakan untuk membuat model.

Salah satu cara untuk mengatasi masalah ini adalah dengan membagi data lagi, jadi kami memiliki tiga set: set pelatihan untuk membangun model, set validasi (atau pengembangan) untuk memilih parameter model, dan set pengujian untuk mengevaluasi kinerja dari parameter yang dipilih. Gambar 5.5 menunjukkan seperti apa ini:

**In[19]:**

```
mglearn.plots.plot_threefold_split()
```



**Gambar 5.5** Pemisahan data tiga kali lipat menjadi set pelatihan, set validasi, dan set pengujian

Setelah memilih parameter terbaik menggunakan set validasi, kami dapat membangun kembali model menggunakan pengaturan parameter yang kami temukan, tetapi sekarang melatih data pelatihan dan data validasi. Dengan cara ini, kita dapat menggunakan data sebanyak mungkin untuk membangun model kita. Ini mengarah pada implementasi berikut:

**In[20]:**

```
from sklearn.svm import SVC
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("Size of training set: {} size of validation set: {} size of test set: {}"
      .format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_valid, y_valid)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

```
# rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

**Out[20]:**

```
Size of training set: 84    size of validation set: 28    size of test set: 38

Best score on validation set: 0.96
Best parameters: {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

Skor terbaik pada set validasi adalah 96%: sedikit lebih rendah dari sebelumnya, mungkin karena kami menggunakan lebih sedikit data untuk melatih model ( $X_{train}$  sekarang lebih kecil karena kami membagi set data dua kali). Namun, skor pada set tes—skor yang sebenarnya memberi tahu kita seberapa baik kita menggeneralisasi—bahkan lebih rendah, yaitu 92%. Jadi kita hanya bisa mengklaim untuk mengklasifikasikan data baru 92% dengan benar, bukan 97% dengan benar seperti yang kita duga sebelumnya!

Perbedaan antara set pelatihan, set validasi, dan set pengujian pada dasarnya penting untuk menerapkan metode pembelajaran mesin dalam praktik. Setiap pilihan yang dibuat berdasarkan informasi "kebocoran" akurasi set tes dari set tes ke dalam model. Oleh karena itu, penting untuk menyimpan satu set tes terpisah, yang hanya digunakan untuk evaluasi akhir. Ini adalah praktik yang baik untuk melakukan semua analisis eksplorasi dan pemilihan model menggunakan kombinasi pelatihan dan set validasi, dan memesan set tes untuk evaluasi akhir—ini bahkan berlaku untuk visualisasi eksplorasi. Sebenarnya, mengevaluasi lebih dari satu model pada set pengujian dan memilih yang lebih baik dari keduanya akan menghasilkan perkiraan yang terlalu optimis tentang seberapa akurat model tersebut.

## 5.5 PENCARIAN KOTAK DENGAN VALIDASI SILANG

Meskipun metode pemisahan data menjadi pelatihan, validasi, dan set pengujian yang baru saja kita lihat dapat diterapkan, dan relatif umum digunakan, metode ini cukup sensitif terhadap bagaimana tepatnya data tersebut dibagi. Dari output cuplikan kode sebelumnya kita dapat melihat bahwa GridSearchCV memilih ' $C$ : 10, ' $\gamma$ ': 0,001 sebagai parameter terbaik, sedangkan output dari kode di bagian sebelumnya memilih ' $C$ : 100, ' $\gamma$ ' : 0,001 sebagai parameter terbaik. Untuk perkiraan kinerja generalisasi yang lebih baik, daripada menggunakan pemisahan tunggal menjadi pelatihan dan set validasi, kita dapat menggunakan validasi silang untuk mengevaluasi kinerja setiap kombinasi parameter. Metode ini dapat dikodekan sebagai berikut:

In[21]:

```

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)

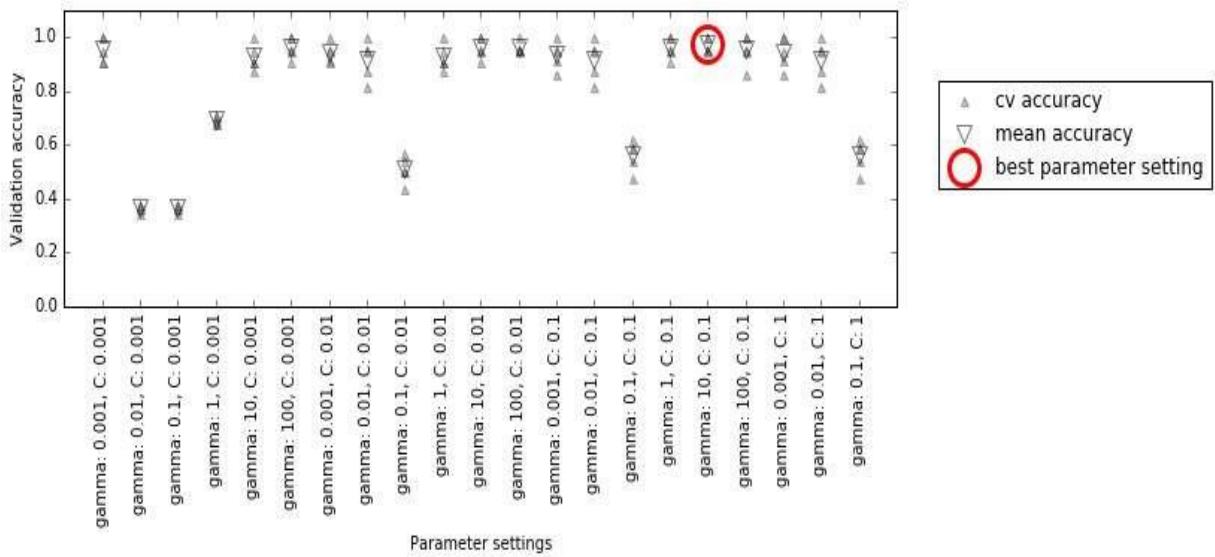
```

Untuk mengevaluasi akurasi SVM menggunakan pengaturan C dan gamma tertentu menggunakan validasi silang lima kali lipat, kita perlu melatih model  $36 * 5 = 180$ . Seperti yang dapat Anda bayangkan, kelemahan utama penggunaan validasi silang adalah waktu yang dibutuhkan untuk melatih semua model ini.

Visualisasi berikut (Gambar 5.6) mengilustrasikan bagaimana pengaturan parameter terbaik dipilih dalam kode sebelumnya:

In[22]:

```
mglearn.plots.plot_cross_val_selection()
```



Gambar 5.6 Hasil pencarian grid dengan validasi silang

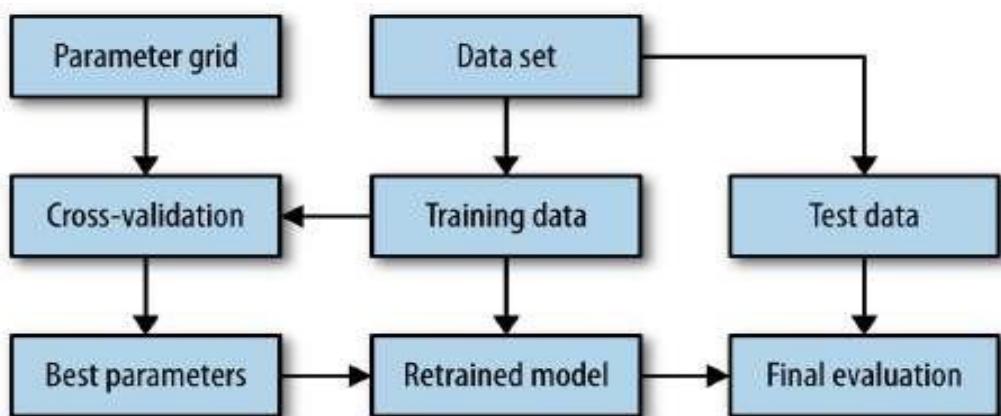
Untuk setiap pengaturan parameter (hanya sebagian yang ditampilkan), lima nilai akurasi dihitung, satu untuk setiap pemisahan dalam validasi silang. Kemudian akurasi validasi rata-rata dihitung untuk setiap pengaturan parameter. Parameter dengan akurasi validasi rata-rata tertinggi dipilih, ditandai dengan lingkaran.



Seperti yang kami katakan sebelumnya, validasi silang adalah cara untuk mengevaluasi algoritma yang diberikan pada kumpulan data tertentu. Namun, sering digunakan bersama dengan metode pencarian parameter seperti pencarian grid. Untuk alasan ini, banyak orang menggunakan istilah validasi silang bahasa sehari-hari untuk merujuk pada pencarian kisi dengan validasi silang. Keseluruhan proses pemisahan data, menjalankan pencarian grid, dan mengevaluasi parameter akhir diilustrasikan pada Gambar 5.7:

**In[23]:**

```
mlearn.plots.plot_grid_search_overview()
```



**Gambar 5.7** Gambaran umum proses pemilihan parameter dan evaluasi model dengan GridSearchCV

Karena pencarian grid dengan validasi silang adalah metode yang umum digunakan untuk menyesuaikan parameter, scikit-learn menyediakan kelas GridSearchCV, yang mengimplementasikannya dalam bentuk estimator. Untuk menggunakan kelas GridSearchCV, Anda harus terlebih dahulu menentukan parameter yang ingin Anda cari menggunakan kamus. GridSearchCV kemudian akan melakukan semua model yang diperlukan. Kunci kamus adalah nama parameter yang ingin kita sesuaikan (seperti yang diberikan saat membangun model—dalam hal ini, C dan gamma), dan nilainya adalah pengaturan parameter yang ingin kita coba. Mencoba nilai 0,001, 0,01, 0,1, 1, 10, dan 100 untuk C dan gamma diterjemahkan ke kamus berikut:

**In[24]:**

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_grid))
```

**Out[24]:**

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

Sekarang kita dapat membuat instance kelas GridSearchCV dengan model (SVC), parameter grid yang akan dicari (param\_grid), dan strategi validasi silang yang ingin kita gunakan (misalnya, validasi silang bertingkat lima kali lipat):

In[25]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
```

GridSearchCV akan menggunakan validasi silang sebagai pengganti pemisahan menjadi set pelatihan dan validasi yang kita gunakan sebelumnya. Namun, kita masih perlu membagi data menjadi set pelatihan dan pengujian, untuk menghindari parameter yang berlebihan:

In[26]:

```
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
```

Objek grid\_search yang kita buat berperilaku seperti classifier; kita dapat memanggil metode standar fit, predict, dan score di atasnya.<sup>1</sup> Namun, ketika kita memanggil fit, itu akan menjalankan validasi silang untuk setiap kombinasi parameter yang kita tentukan di param\_grid:

In[27]:

```
grid_search.fit(X_train, y_train)
```

Memasang objek GridSearchCV tidak hanya mencari parameter terbaik, tetapi juga secara otomatis menyesuaikan model baru di seluruh dataset pelatihan dengan parameter yang menghasilkan kinerja validasi silang terbaik. Apa yang terjadi di fit oleh karena itu setara dengan hasil dari kode In[21] yang kita lihat di awal bagian ini. Kelas GridSearchCV menyediakan antarmuka yang sangat nyaman untuk mengakses model yang dilatih ulang menggunakan metode prediksi dan skor. Untuk mengevaluasi seberapa baik parameter yang ditemukan digeneralisasi, kita dapat memanggil skor pada set tes:

In[28]:

```
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

Out[28]:

```
Test set score: 0.97
```

Memilih parameter menggunakan validasi silang, kami benar-benar menemukan model yang mencapai akurasi 97% pada set pengujian. Yang penting di sini adalah kita tidak menggunakan test set untuk memilih parameter. Parameter yang ditemukan diberi skor dalam atribut best\_params\_, dan akurasi validasi silang terbaik (akurasi rata-rata pada pemisahan berbeda untuk pengaturan parameter ini) disimpan di best\_score\_:

In[29]:

```
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

Out[29]:

```
Best parameters: {'C': 100, 'gamma': 0.01}
Best cross-validation score: 0.97
```



Sekali lagi, berhati-hatilah untuk tidak mengacaukan `best_score_` dengan kinerja generalisasi model yang dihitung dengan metode skor pada set tes. Menggunakan metode skor (atau mengevaluasi output dari metode prediksi) menggunakan model yang dilatih di seluruh rangkaian pelatihan.

Atribut `best_score_` menyimpan akurasi validasi silang rata-rata, dengan validasi silang dilakukan pada set pelatihan.

Terkadang sangat membantu untuk memiliki akses ke model aktual yang ditemukan—misalnya, untuk melihat koefisien atau fitur penting. Anda dapat mengakses model dengan parameter terbaik yang dilatih di seluruh rangkaian pelatihan menggunakan atribut `best_estimator_`:

**In[30]:**

```
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

**Out[30]:**

```
Best estimator:
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Karena `grid_search` sendiri memiliki metode prediksi dan skor, menggunakan `best_estimator_` tidak diperlukan untuk membuat prediksi atau mengevaluasi model.

### Menganalisis hasil validasi silang

Seringkali membantu untuk memvisualisasikan hasil validasi silang, untuk memahami bagaimana generalisasi model bergantung pada parameter yang kita cari. Karena pencarian grid cukup mahal secara komputasi untuk dijalankan, seringkali merupakan ide yang baik untuk memulai dengan grid yang relatif kasar dan kecil. Kami kemudian dapat memeriksa hasil pencarian grid yang divalidasi silang, dan mungkin memperluas pencarian kami. Hasil pencarian grid dapat ditemukan di atribut `cv_results_`, yang merupakan kamus yang menyimpan semua aspek pencarian. Ini berisi banyak detail, seperti yang Anda lihat di output berikut, dan paling baik dilihat setelah mengonversinya menjadi pandas DataFrame:

**In[31]:**

```
import pandas as pd
# convert to DataFrame
results = pd.DataFrame(grid_search.cv_results_)
# show the first 5 rows
display(results.head())
```

**Out[31]:**

	param_C	param_gamma	params	mean_test_score
0	0.001	0.001	{'C': 0.001, 'gamma': 0.001}	0.366
1	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	0.366
2	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	0.366
3	0.001	1	{'C': 0.001, 'gamma': 1}	0.366
4	0.001	10	{'C': 0.001, 'gamma': 10}	0.366

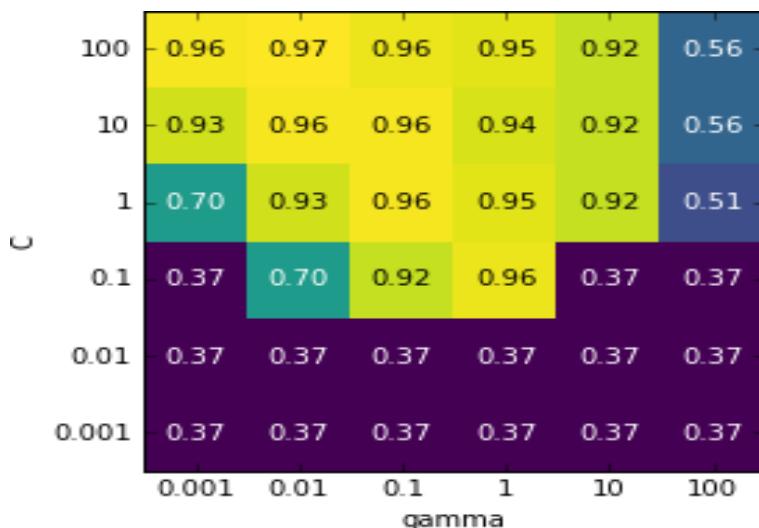
	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	22	0.375	0.347	0.363
1	22	0.375	0.347	0.363
2	22	0.375	0.347	0.363
3	22	0.375	0.347	0.363
4	22	0.375	0.347	0.363
	split3_test_score	split4_test_score	std_test_score	
0	0.363	0.380	0.011	
1	0.363	0.380	0.011	
2	0.363	0.380	0.011	
3	0.363	0.380	0.011	
4	0.363	0.380	0.011	

Setiap baris dalam hasil sesuai dengan satu pengaturan parameter tertentu. Untuk setiap pengaturan, hasil dari semua pemisahan validasi silang dicatat, serta rata-rata dan simpangan baku untuk semua pemisahan. Saat kami mencari parameter grid dua dimensi (C dan gamma), ini paling baik divisualisasikan sebagai peta panas (Gambar 5.8). Pertama-tama kita ekstrak skor validasi rata-rata, kemudian kita bentuk kembali skor sehingga sumbu sesuai dengan C dan gamma:

In[32]:

```
scores = np.array(results.mean_test_score).reshape(6, 6)

# plot the mean cross-validation scores
mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=param_grid['gamma'],
                      ylabel='C', yticklabels=param_grid['C'], cmap="viridis")
```

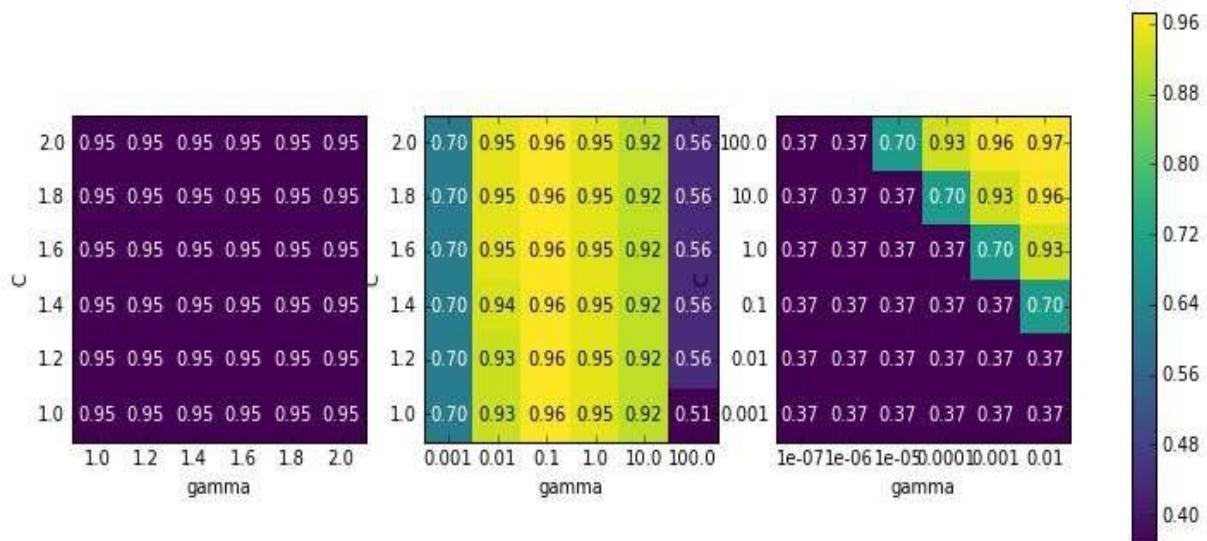


Gambar 5.8 Peta panas skor validasi silang rata-rata sebagai fungsi C dan gamma

Setiap titik di peta panas sesuai dengan satu kali validasi silang, dengan pengaturan parameter tertentu. Warna mengkodekan akurasi validasi silang, dengan warna terang berarti akurasi tinggi dan warna gelap berarti akurasi rendah. Anda dapat melihat bahwa SVC sangat sensitif terhadap pengaturan parameter. Untuk banyak pengaturan parameter, akurasinya sekitar 40%, yang cukup buruk; untuk pengaturan lain akurasinya sekitar 96%. Kita dapat mengambil dari plot ini beberapa hal. Pertama, parameter yang kami sesuaikan sangat penting untuk mendapatkan kinerja yang baik. Kedua parameter (C dan gamma) sangat

penting, karena menyesuaikannya dapat mengubah akurasi dari 40% menjadi 96%. Selain itu, rentang yang kami pilih untuk parameter adalah rentang di mana kami melihat perubahan signifikan dalam hasil. Penting juga untuk dicatat bahwa rentang untuk parameter cukup besar: nilai optimal untuk setiap parameter tidak berada di tepi plot.

Sekarang mari kita lihat beberapa plot (ditunjukkan pada Gambar 5.9) di mana hasilnya kurang ideal, karena rentang pencarian tidak dipilih dengan benar:



**Gambar 5.9** Visualisasi peta panas dari kisi pencarian yang salah ditentukan

In[33]:

```
fig, axes = plt.subplots(1, 3, figsize=(13, 5))

param_grid_linear = {'C': np.linspace(1, 2, 6),
                     'gamma': np.linspace(1, 2, 6)}

param_grid_one_log = {'C': np.linspace(1, 2, 6),
                      'gamma': np.logspace(-3, 2, 6)}

param_grid_range = {'C': np.logspace(-3, 2, 6),
                    'gamma': np.logspace(-7, -2, 6)}

for param_grid, ax in zip([param_grid_linear, param_grid_one_log,
                           param_grid_range], axes):
    grid_search = GridSearchCV(SVC(), param_grid, cv=5)
    grid_search.fit(X_train, y_train)
    scores = grid_search.cv_results_['mean_test_score'].reshape(6, 6)

    # plot the mean cross-validation scores
    scores_image = mglearn.tools.heatmap(
        scores, xlabel='gamma', ylabel='C', xticklabels=param_grid['gamma'],
        yticklabels=param_grid['C'], cmap="viridis", ax=ax)

    plt.colorbar(scores_image, ax=axes.tolist())
```

Panel pertama tidak menunjukkan perubahan sama sekali, dengan warna konstan di seluruh grid parameter. Dalam hal ini, ini disebabkan oleh penskalaan dan rentang parameter C dan gamma yang tidak tepat. Namun, jika tidak ada perubahan dalam akurasi yang terlihat pada pengaturan parameter yang berbeda, bisa jadi sebuah parameter tidak penting sama

sekali. Biasanya baik untuk mencoba nilai yang sangat ekstrim terlebih dahulu, untuk melihat apakah ada perubahan akurasi sebagai akibat dari perubahan parameter.

Panel kedua menunjukkan pola garis vertikal. Ini menunjukkan bahwa hanya pengaturan parameter gamma yang membuat perbedaan. Ini bisa berarti bahwa parameter gamma mencari nilai yang menarik tetapi parameter C tidak—atau bisa berarti parameter C tidak penting.

Panel ketiga menunjukkan perubahan pada C dan gamma. Namun, kita dapat melihat bahwa di seluruh kiri bawah plot, tidak ada hal menarik yang terjadi. Kami mungkin dapat mengecualikan nilai yang sangat kecil dari penelusuran kisi di masa mendatang. Pengaturan parameter optimal ada di kanan atas. Karena yang optimal ada di perbatasan plot, kita dapat berharap bahwa mungkin ada nilai yang lebih baik lagi di luar batas ini, dan kita mungkin ingin mengubah rentang pencarian untuk memasukkan lebih banyak parameter di wilayah ini.

Menyetel kisi parameter berdasarkan skor validasi silang sangat baik, dan cara yang baik untuk mengeksplorasi pentingnya parameter yang berbeda. Namun, Anda tidak boleh menguji rentang parameter yang berbeda pada set pengujian akhir—seperti yang telah kita bahas sebelumnya, evaluasi set pengujian harus dilakukan hanya setelah kita tahu persis model apa yang ingin kita gunakan.

### Telusuri ruang yang bukan kisi

Dalam beberapa kasus, mencoba semua kemungkinan kombinasi dari semua parameter seperti yang biasa dilakukan GridSearchCV, bukanlah ide yang baik. Misalnya, SVC memiliki parameter kernel, dan tergantung pada kernel mana yang dipilih, parameter lain akan relevan. Jika ker nel='linier', modelnya linier, dan hanya parameter C yang digunakan. Jika kernel='rbf', parameter C dan gamma digunakan (tetapi bukan parameter lain seperti derajat). Dalam hal ini, mencari semua kemungkinan kombinasi C, gamma, dan kernel tidak akan masuk akal: jika kernel='linier', gamma tidak digunakan, dan mencoba nilai gamma yang berbeda akan membuang-buang waktu. Untuk menangani parameter "bersyarat" semacam ini, GridSearchCV memungkinkan param\_grid menjadi daftar kamus. Setiap kamus dalam daftar diperluas menjadi kisi independen. Kemungkinan pencarian grid yang melibatkan kernel dan parameter dapat terlihat seperti ini:

**In[34]:**

```
param_grid = [{"kernel": ['rbf'],
   'C': [0.001, 0.01, 0.1, 1, 10, 100],
   'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
  {"kernel": ['linear'],
   'C': [0.001, 0.01, 0.1, 1, 10, 100]}]
print("List of grids:\n{}".format(param_grid))
```

**Out[34]:**

```
List of grids:
[{"kernel": ['rbf'], 'C': [0.001, 0.01, 0.1, 1, 10, 100],
 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
 {"kernel": ['linear'], 'C': [0.001, 0.01, 0.1, 1, 10, 100]}]
```

Di grid pertama, parameter kernel selalu disetel ke 'rbf' (bukan berarti entri untuk kernel adalah daftar panjang satu), dan parameter C dan gamma bervariasi. Di grid kedua, parameter kernel selalu diatur ke linier, dan hanya C yang bervariasi. Sekarang mari kita terapkan pencarian parameter yang lebih kompleks ini:

**In[35]:**

```
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

**Out[35]:**

```
Best parameters: {'C': 100, 'kernel': 'rbf', 'gamma': 0.01}
Best cross-validation score: 0.97
```

Mari kita lihat cv\_results\_ lagi. Seperti yang diharapkan, jika kernel 'linier', maka hanya C yang bervariasi:

**In[36]:**

```
results = pd.DataFrame(grid_search.cv_results_)
# we display the transposed table so that it better fits on the page:
display(results.T)
```

**Out[36]:**

	0	1	2	3	...	38	39	40	41
param_C	0.001	0.001	0.001	0.001	...	0.1	1	10	100
param_gamma	0.001	0.01	0.1	1	...	NaN	NaN	NaN	NaN
param_kernel	rbf	rbf	rbf	rbf	...	linear	linear	linear	linear
params	{C: 0.001, kernel: rbf, gamma: 0.001}	{C: 0.001, kernel: rbf, gamma: 0.01}	{C: 0.001, kernel: rbf, gamma: 0.1}	{C: 0.001, kernel: rbf, gamma: 1}	...	{C: 0.1, kernel: linear}	{C: 1, kernel: linear}	{C: 10, kernel: linear}	{C: 100, kernel: linear}
mean_test_score	0.37	0.37	0.37	0.37	...	0.95	0.97	0.96	0.96
rank_test_score	27	27	27	27	...	11	1	3	3
split0_test_score	0.38	0.38	0.38	0.38	...	0.96	1	0.96	0.96
split1_test_score	0.35	0.35	0.35	0.35	...	0.91	0.96	1	1
split2_test_score	0.36	0.36	0.36	0.36	...	1	1	1	1
split3_test_score	0.36	0.36	0.36	0.36	...	0.91	0.95	0.91	0.91
split4_test_score	0.38	0.38	0.38	0.38	...	0.95	0.95	0.95	0.95
std_test_score	0.011	0.011	0.011	0.011	...	0.033	0.022	0.034	0.034

12 baris × 42 kolom

### Menggunakan strategi validasi silang yang berbeda dengan pencarian grid

Sama halnya dengan cross\_val\_score, GridSearchCV menggunakan validasi silang k-fold bertingkat secara default untuk klasifikasi, dan validasi silang k-fold untuk regresi. Namun, Anda juga dapat melewatkannya pembagi validasi silang apa pun, seperti yang dijelaskan di "Kontrol lebih banyak atas validasi silang" di halaman 256, sebagai parameter cv di GridSearchCV. Khususnya, untuk mendapatkan hanya satu pemisahan menjadi satu set

pelatihan dan validasi, Anda dapat menggunakan ShuffleSplit atau StratifiedShuffleSplit dengan n\_iter=1. Ini mungkin berguna untuk kumpulan data yang sangat besar, atau model yang sangat lambat.

### Validasi silang bersarang

Dalam contoh sebelumnya, kami beralih dari menggunakan pemisahan tunggal data menjadi pelatihan, validasi, dan set pengujian menjadi membagi data menjadi set pelatihan dan pengujian, kemudian melakukan validasi silang pada set pelatihan. Tetapi ketika menggunakan GridSearchCV seperti yang dijelaskan sebelumnya, kami masih memiliki satu pemisahan data menjadi set pelatihan dan pengujian, yang mungkin membuat hasil kami tidak stabil dan membuat kami terlalu bergantung pada pemisahan data tunggal ini. Kita dapat melangkah lebih jauh, dan alih-alih membagi data asli menjadi set pelatihan dan pengujian satu kali, gunakan beberapa pemisahan validasi silang. Ini akan menghasilkan apa yang disebut validasi silang bersarang. Dalam validasi silang bersarang, ada loop luar di atas pemisahan data menjadi set pelatihan dan pengujian. Untuk masing-masing dari mereka, pencarian grid dijalankan (yang mungkin menghasilkan parameter terbaik yang berbeda untuk setiap pemisahan di loop luar). Kemudian, untuk setiap split luar, skor set tes menggunakan pengaturan terbaik dilaporkan.

Hasil dari prosedur ini adalah daftar skor—bukan model, dan bukan pengaturan parameter. Skor memberi tahu kita seberapa baik model menggeneralisasi, mengingat parameter terbaik yang ditemukan oleh grid. Karena tidak menyediakan model yang dapat digunakan pada data baru, validasi silang bersarang jarang digunakan saat mencari model prediktif untuk diterapkan pada data di masa mendatang. Namun, ini dapat berguna untuk mengevaluasi seberapa baik model tertentu bekerja pada kumpulan data tertentu.

Menerapkan validasi silang bersarang di scikit-learn sangatlah mudah. Kami memanggil cross\_val\_score dengan instance GridSearchCV sebagai model:

**In[34]:**

```
scores = cross_val_score(GridSearchCV(SVC(), param_grid, cv=5),
                        iris.data, iris.target, cv=5)
print("Cross-validation scores: ", scores)
print("Mean cross-validation score: ", scores.mean())
```

**Out[34]:**

```
Cross-validation scores: [ 0.967  1.      0.967  0.967  1.      ]
Mean cross-validation score:  0.98
```

Hasil validasi silang bersarang kami dapat diringkas sebagai "SVC dapat mencapai akurasi validasi silang rata-rata 98% pada set data iris"—tidak lebih dan tidak kurang.

Di sini, kami menggunakan validasi silang lima kali lipat bertingkat di loop dalam dan luar. Karena param\_grid kami berisi 36 kombinasi parameter, ini menghasilkan  $36 * 5 * 5 = 900$  model yang sedang dibangun, membuat validasi silang bersarang menjadi prosedur yang sangat mahal. Di sini, kami menggunakan pembagi validasi silang yang sama di loop dalam dan luar; namun, ini tidak perlu dan Anda dapat menggunakan kombinasi strategi validasi silang apa pun di loop dalam dan luar. Mungkin agak sulit untuk memahami apa yang terjadi dalam

satu baris yang diberikan di atas, dan akan sangat membantu untuk memvisualisasikannya sebagai perulangan, seperti yang dilakukan dalam implementasi sederhana berikut:

**In[35]:**

```
def nested_cv(X, y, inner_cv, outer_cv, Classifier, parameter_grid):
    outer_scores = []
    # for each split of the data in the outer cross-validation
    # (split method returns indices)
    for training_samples, test_samples in outer_cv.split(X, y):
        # find best parameter using inner cross-validation
        best_parms = {}
        best_score = -np.inf
        # iterate over parameters
        for parameters in parameter_grid:
            # accumulate score over inner splits
            cv_scores = []
            # iterate over inner cross-validation
            for inner_train, inner_test in inner_cv.split(
                X[training_samples], y[training_samples]):
                # build classifier given parameters and training data
                clf = Classifier(**parameters)
                clf.fit(X[inner_train], y[inner_train])
                # evaluate on inner test set
                score = clf.score(X[inner_test], y[inner_test])
                cv_scores.append(score)
            # compute mean score over inner folds
            mean_score = np.mean(cv_scores)
            if mean_score > best_score:
                # if better than so far, remember parameters
                best_score = mean_score
                best_params = parameters
            # build classifier on best parameters using outer training set
            clf = Classifier(**best_params)
            clf.fit(X[training_samples], y[training_samples])
            # evaluate
            outer_scores.append(clf.score(X[test_samples], y[test_samples]))
    return np.array(outer_scores)
```

Sekarang, mari kita jalankan fungsi ini pada dataset iris:

**In[36]:**

```
from sklearn.model_selection import ParameterGrid, StratifiedKFold
scores = nested_cv(iris.data, iris.target, StratifiedKFold(5),
                    StratifiedKFold(5), SVC, ParameterGrid(param_grid))
print("Cross-validation scores: {}".format(scores))
```

**Out[36]:**

```
Cross-validation scores: [ 0.967  1.      0.967  0.967  1.      ]
```

### Memparalelkan validasi silang dan pencarian grid

Saat menjalankan pencarian grid pada banyak parameter dan pada kumpulan data yang besar dapat menjadi tantangan komputasi, itu juga paralel. Ini berarti bahwa membangun model menggunakan pengaturan parameter tertentu pada pemisahan validasi silang tertentu dapat dilakukan sepenuhnya secara independen dari pengaturan parameter dan model lainnya. Hal ini membuat pencarian grid dan validasi silang menjadi kandidat ideal untuk paralelisasi pada beberapa inti CPU atau pada sebuah cluster. Anda dapat menggunakan beberapa inti di Grid SearchCV dan cross\_val\_score dengan mengatur parameter n\_jobs ke

jumlah inti CPU yang ingin Anda gunakan. Anda dapat mengatur `n_jobs=-1` untuk menggunakan semua inti yang tersedia.

Anda harus menyadari bahwa scikit-learn tidak mengizinkan operasi paralel bersarang. Jadi, jika Anda menggunakan opsi `n_jobs` pada model Anda (misalnya, hutan acak), Anda tidak dapat menggunakan di `GridSearchCV` untuk mencari model ini. Jika set data dan model Anda sangat besar, mungkin penggunaan banyak core menghabiskan terlalu banyak memori, dan Anda harus memantau penggunaan memori saat membuat model besar secara paralel.

Dimungkinkan juga untuk memparalelkan pencarian grid dan validasi silang pada beberapa mesin dalam sebuah cluster, meskipun pada saat penulisan ini tidak didukung dalam scikit-learn. Namun, dimungkinkan untuk menggunakan kerangka kerja paralel IPython untuk pencarian grid paralel, jika Anda tidak keberatan menulis parameter loop over seperti yang kita lakukan di “Pencarian Grid Sederhana”.

Untuk pengguna Spark, ada juga paket `spark-sklearn` yang dikembangkan baru-baru ini, yang memungkinkan menjalankan pencarian grid di atas cluster Spark yang sudah ada.

## 5.6 MATRIK EVALUASI DAN SKOR

Sejauh ini, kami telah mengevaluasi kinerja klasifikasi menggunakan akurasi (fraksi sampel yang diklasifikasikan dengan benar) dan kinerja regresi menggunakan R2. Namun, ini hanya dua dari banyak cara yang mungkin untuk meringkas seberapa baik kinerja model yang diawasi pada kumpulan data yang diberikan. Dalam praktiknya, metrik evaluasi ini mungkin tidak sesuai untuk aplikasi Anda, dan penting untuk memilih metrik yang tepat saat memilih antara model dan menyesuaikan parameter.

### *Tetap Ingat Tujuan Akhir*

Saat memilih metrik, Anda harus selalu mempertimbangkan tujuan akhir dari aplikasi pembelajaran mesin. Dalam praktiknya, kita biasanya tidak hanya tertarik untuk membuat prediksi yang akurat, tetapi juga menggunakan prediksi ini sebagai bagian dari proses pengambilan keputusan yang lebih besar. Sebelum memilih metrik pembelajaran mesin, Anda harus memikirkan tujuan aplikasi tingkat tinggi, yang sering disebut metrik bisnis. Konsekuensi dari memilih algoritme tertentu untuk aplikasi pembelajaran mesin disebut dampak bisnis.<sup>2</sup> Mungkin tujuan tingkat tinggi adalah menghindari kecelakaan lalu lintas, atau mengurangi jumlah rawat inap di rumah sakit. Itu juga bisa mendapatkan lebih banyak pengguna untuk situs web Anda, atau membuat pengguna menghabiskan lebih banyak uang di toko Anda. Saat memilih model atau menyesuaikan parameter, Anda harus memilih model atau nilai parameter yang memiliki pengaruh paling positif pada metrik bisnis. Seringkali ini sulit, karena menilai dampak bisnis dari model tertentu mungkin memerlukan memasukkannya ke dalam produksi dalam sistem kehidupan nyata.

Pada tahap awal pengembangan, dan untuk menyesuaikan parameter, seringkali tidak mungkin untuk memasukkan model ke dalam produksi hanya untuk tujuan pengujian, karena risiko bisnis atau pribadi yang tinggi yang dapat terlibat. Bayangkan mengevaluasi kemampuan menghindari pejalan kaki dari mobil self-driving hanya dengan membiarkannya mengemudi, tanpa memverifikasinya terlebih dahulu; jika model Anda buruk, pejalan kaki akan mendapat masalah! Oleh karena itu kita sering perlu menemukan beberapa prosedur evaluasi pengganti,

menggunakan metrik evaluasi yang lebih mudah untuk dihitung. Misalnya, kami dapat menguji pengklasifikasian gambar pejalan kaki terhadap non-pejalan kaki dan mengukur akurasi. Ingatlah bahwa ini hanya pengganti, dan akan terbayar untuk menemukan metrik terdekat dengan tujuan bisnis awal yang layak untuk dievaluasi. Metrik terdekat ini harus digunakan bila memungkinkan untuk evaluasi dan pemilihan model. Hasil evaluasi ini mungkin tidak berupa angka tunggal—konsekuensi algoritme Anda bisa jadi Anda memiliki 10% lebih banyak pelanggan, tetapi setiap pelanggan akan membelanjakan 15% lebih sedikit—tetapi harus menangkap dampak bisnis yang diharapkan dari memilih satu model di atas yang lain. Pada bagian ini, pertama-tama kita akan membahas metrik untuk kasus khusus yang penting dari klasifikasi biner, kemudian beralih ke klasifikasi multikelas dan akhirnya regresi.

## 5.7 METRIK UNTUK KLASIFIKASI BINER

Klasifikasi biner bisa dibilang aplikasi pembelajaran mesin yang paling umum dan sederhana secara konseptual dalam praktiknya. Namun, masih ada sejumlah peringatan dalam mengevaluasi tugas sederhana ini. Sebelum kita mendalami metrik alternatif, mari kita lihat cara-cara di mana akurasi pengukuran bisa menyesatkan. Ingatlah bahwa untuk klasifikasi biner, kita sering berbicara tentang kelas positif dan kelas negatif, dengan pemahaman bahwa kelas positif adalah yang kita cari.

### Jenis kesalahan

Seringkali, akurasi bukanlah ukuran kinerja prediksi yang baik, karena jumlah kesalahan yang kita buat tidak memuat semua informasi yang kita minati. Bayangkan sebuah aplikasi untuk menyaring deteksi dini kanker menggunakan tes otomatis. Jika tes negatif, pasien akan dianggap sehat, sedangkan jika tes positif, pasien akan menjalani pemeriksaan tambahan. Di sini, kita akan menyebut tes positif (indikasi kanker) sebagai kelas positif, dan tes negatif sebagai kelas negatif. Kami tidak dapat berasumsi bahwa model kami akan selalu bekerja dengan sempurna, dan itu akan membuat kesalahan. Untuk aplikasi apa pun, kita perlu bertanya pada diri sendiri apa konsekuensi dari kesalahan ini di dunia nyata.

Satu kesalahan yang mungkin adalah bahwa pasien yang sehat akan diklasifikasikan sebagai positif, yang mengarah ke pengujian tambahan. Hal ini menyebabkan beberapa biaya dan ketidaknyamanan bagi pasien (dan mungkin beberapa tekanan mental). Prediksi positif yang salah disebut positif palsu. Kemungkinan kesalahan lainnya adalah pasien yang sakit akan diklasifikasikan sebagai negatif, dan tidak akan menerima tes dan perawatan lebih lanjut. Kanker yang tidak terdiagnosis dapat menyebabkan masalah kesehatan yang serius, dan bahkan bisa berakibat fatal. Kesalahan semacam ini—prediksi negatif yang salah—disebut negatif palsu. Dalam statistik, positif palsu juga dikenal sebagai kesalahan tipe I, dan negatif palsu sebagai kesalahan tipe II. Kami akan tetap berpegang pada "negatif palsu" dan "positif palsu", karena lebih eksplisit dan lebih mudah diingat. Dalam contoh diagnosis kanker, jelas bahwa kita ingin menghindari negatif palsu sebanyak mungkin, sementara positif palsu dapat dipandang sebagai gangguan kecil.

Meskipun ini adalah contoh yang sangat drastis, konsekuensi dari positif palsu dan negatif palsu jarang sama. Dalam aplikasi komersial, dimungkinkan untuk menetapkan nilai dolar untuk kedua jenis kesalahan, yang memungkinkan pengukuran kesalahan prediksi

tertentu dalam dolar, bukan akurasi. Ini mungkin jauh lebih bermakna untuk membuat keputusan bisnis tentang model mana yang akan digunakan.

### Kumpulan data tidak seimbang

Jenis kesalahan memainkan peran penting ketika salah satu dari dua kelas jauh lebih sering daripada yang lain. Ini sangat umum dalam praktik; contoh yang baik adalah prediksi klik-tayang, di mana setiap titik data mewakili "tayangan", item yang ditampilkan kepada pengguna. Item ini mungkin iklan, atau cerita terkait, atau orang terkait untuk diikuti di situs media sosial. Tujuannya adalah untuk memprediksi apakah, jika ditampilkan item tertentu, pengguna akan mengkliknya (menunjukkan bahwa mereka tertarik). Sebagian besar hal yang ditampilkan pengguna di Internet (khususnya, iklan) tidak akan menghasilkan klik. Anda mungkin perlu menampilkan 100 iklan atau artikel kepada pengguna sebelum mereka menemukan sesuatu yang cukup menarik untuk diklik. Ini menghasilkan kumpulan data di mana untuk setiap 99 titik data "tidak ada klik", ada 1 titik data yang "diklik"; dengan kata lain, 99% sampel termasuk dalam kelas "tidak ada klik". Kumpulan data di mana satu kelas jauh lebih sering daripada yang lain sering disebut kumpulan data tidak seimbang, atau kumpulan data dengan kelas tidak seimbang. Pada kenyataannya, data yang tidak seimbang adalah norma, dan jarang terjadi peristiwa yang menarik memiliki frekuensi yang sama atau bahkan serupa dalam data.

Sekarang katakanlah Anda membuat pengklasifikasi yang 99% akurat pada tugas prediksi klik. Apa artinya itu? Akurasi 99% terdengar mengesankan, tetapi ini tidak memperhitungkan ketidakseimbangan kelas. Anda dapat mencapai akurasi 99% tanpa membuat model pembelajaran mesin, dengan selalu memprediksi "tidak ada klik". Di sisi lain, bahkan dengan data yang tidak seimbang, model yang akurat 99% sebenarnya cukup bagus. Namun, akurasi tidak memungkinkan kita untuk membedakan model konstan "tanpa klik" dari model yang berpotensi bagus.

Untuk mengilustrasikannya, kami akan membuat kumpulan data tidak seimbang 9:1 dari kumpulan data digit, dengan mengklasifikasikan angka 9 terhadap sembilan kelas lainnya:

**In[37]:**

```
from sklearn.datasets import load_digits

digits = load_digits()
y = digits.target == 9

X_train, X_test, y_train, y_test = train_test_split(
    digits.data, y, random_state=0)
```

Kita dapat menggunakan DummyClassifier untuk selalu memprediksi kelas mayoritas (di sini "bukan sembilan") untuk melihat bagaimana akurasi yang tidak informatif:

**In[38]:**

```
from sklearn.dummy import DummyClassifier
dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
pred_most_frequent = dummy_majority.predict(X_test)
print("Unique predicted labels: {}".format(np.unique(pred_most_frequent)))
print("Test score: {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

**Out[38]:**

```
Unique predicted labels: [False]
Test score: 0.90
```

Kami memperoleh akurasi hampir 90% tanpa mempelajari apa pun. Ini mungkin tampak mencolok, tetapi pikirkanlah sejenak. Bayangkan seseorang memberi tahu Anda bahwa model mereka 90% akurat. Anda mungkin berpikir mereka melakukan pekerjaan yang sangat baik. Tetapi tergantung pada masalahnya, itu mungkin hanya dengan memprediksi satu kelas! Mari kita bandingkan ini dengan menggunakan classifier yang sebenarnya:

**In[39]:**

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
pred_tree = tree.predict(X_test)
print("Test score: {:.2f}".format(tree.score(X_test, y_test)))
```

**Out[39]:**

```
Test score: 0.92
```

Menurut akurasi, DecisionTreeClassifier hanya sedikit lebih baik daripada prediktor konstan. Ini dapat menunjukkan bahwa ada yang salah dengan cara kami menggunakan Decision Tree Classifier, atau bahwa akurasi sebenarnya bukan ukuran yang baik di sini.

Untuk tujuan perbandingan, mari evaluasi dua pengklasifikasi lagi, LogisticRegression dan DummyClassifier default, yang membuat prediksi acak tetapi menghasilkan kelas dengan proporsi yang sama seperti di set pelatihan:

**In[40]:**

```
from sklearn.linear_model import LogisticRegression

dummy = DummyClassifier().fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)
print("dummy score: {:.2f}".format(dummy.score(X_test, y_test)))

logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
pred_logreg = logreg.predict(X_test)
print("logreg score: {:.2f}".format(logreg.score(X_test, y_test)))
```

**Out[40]:**

```
dummy score: 0.80
logreg score: 0.98
```

Pengklasifikasi dummy yang menghasilkan keluaran acak jelas merupakan yang terburuk (menurut akurasi), sedangkan LogisticRegression menghasilkan hasil yang sangat baik. Namun, bahkan pengklasifikasi acak menghasilkan akurasi lebih dari 80%. Hal ini membuat sangat sulit untuk menilai mana dari hasil ini yang benar-benar membantu. Masalahnya di sini adalah bahwa akurasi adalah ukuran yang tidak memadai untuk mengukur kinerja prediktif dalam pengaturan yang tidak seimbang ini. Untuk sisanya ini, kita akan mengeksplorasi metrik alternatif yang memberikan panduan yang lebih baik dalam memilih model. Secara khusus, kami ingin memiliki metrik yang memberi tahu kami seberapa jauh

lebih baik sebuah model daripada membuat prediksi "paling sering" atau prediksi acak, karena dihitung dalam `pred_most_frequent` dan `pred_dummy`. Jika kita menggunakan metrik untuk menilai model kita, itu pasti bisa menyingkirkan prediksi yang tidak masuk akal ini.

### Matriks kebingungan

Salah satu cara paling komprehensif untuk merepresentasikan hasil evaluasi klasifikasi biner adalah dengan menggunakan matriks konfusi. Mari kita periksa prediksi LogisticRegression dari bagian sebelumnya menggunakan fungsi `confusion_matrix`. Kami sudah menyimpan prediksi pada set tes di `pred_logreg`:

**In[41]:**

```
from sklearn.metrics import confusion_matrix

confusion = confusion_matrix(y_test, pred_logreg)
print("Confusion matrix:\n{}".format(confusion))
```

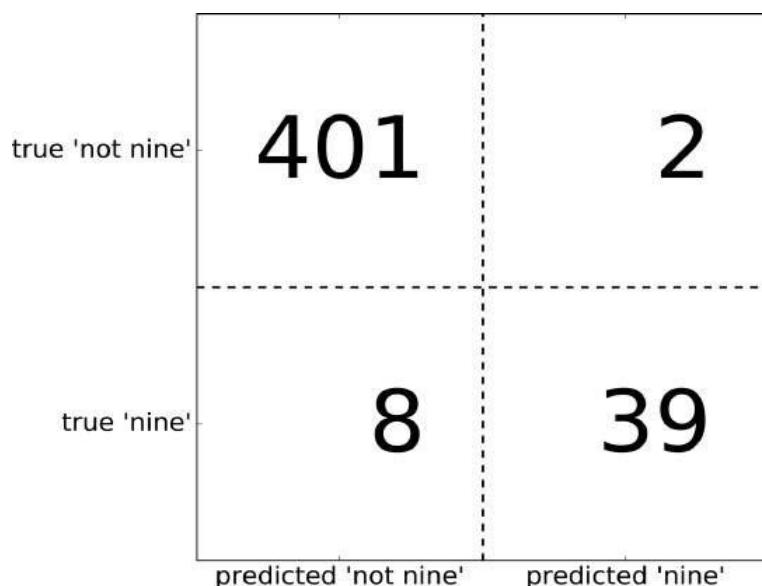
**Out[41]:**

```
Confusion matrix:
[[401  2]
 [ 8 39]]
```

Keluaran dari `confusion_matrix` adalah larik dua kali dua, di mana baris sesuai dengan kelas sebenarnya dan kolom sesuai dengan kelas yang diprediksi. Setiap entri menghitung seberapa sering sampel milik kelas yang sesuai dengan baris (di sini, "bukan sembilan" dan "sembilan") diklasifikasikan sebagai kelas yang sesuai dengan kolom. Plot berikut (Gambar 5.10) menggambarkan arti ini:

**In[42]:**

```
mglearn.plots.plot_confusion_matrix_illustration()
```



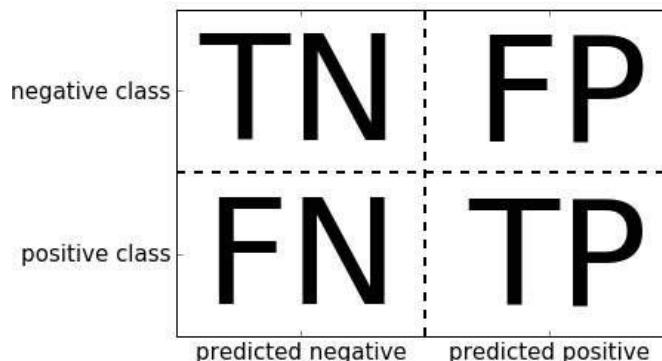
**Gambar 5.10** Matriks kebingungan dari tugas klasifikasi "sembilan vs. istirahat"

Entri pada diagonal utama dari matriks konfusi sesuai dengan klasifikasi yang benar, sementara entri lain memberi tahu kita berapa banyak sampel dari satu kelas yang salah diklasifikasikan sebagai kelas lain.

Jika kita mendeklarasikan “a sembilan” sebagai kelas positif, kita dapat menghubungkan entri dari matriks konfusi dengan istilah positif palsu dan negatif palsu yang telah kita perkenalkan sebelumnya. Untuk melengkapi gambar, kami menyebut sampel yang diklasifikasikan dengan benar milik kelas positif benar-benar positif dan sampel yang diklasifikasikan dengan benar milik kelas negatif benar-benar negatif. Istilah-istilah ini biasanya disingkat FP, FN, TP, dan TN dan mengarah pada interpretasi berikut untuk matriks konfusi (Gambar 5.11):

In[43]:

```
mglearn.plots.plot_binary_confusion_matrix()
```



Gambar 5.11 Matriks kebingungan untuk klasifikasi biner

Sekarang mari kita gunakan matriks konfusi untuk membandingkan model yang kita pasang sebelumnya (dua model dummy, pohon keputusan, dan regresi logistik):

In[44]:

```
print("Most frequent class:")
print(confusion_matrix(y_test, pred_most_frequent))
print("\nDummy model:")
print(confusion_matrix(y_test, pred_dummy))
print("\nDecision tree:")
print(confusion_matrix(y_test, pred_tree))
print("\nLogistic Regression")
print(confusion_matrix(y_test, pred_logreg))
```

**Out[44]:**

```

Most frequent class:
[[403  0]
 [ 47  0]]

Dummy model:
[[361 42]
 [ 43  4]]

Decision tree:
[[390 13]
 [ 24 23]]

Logistic Regression
[[401  2]
 [  8 39]]

```

Melihat matriks kebingungan, cukup jelas bahwa ada sesuatu yang salah dengan `pred_most_frequent`, karena selalu memprediksi kelas yang sama. `pred_dummy`, di sisi lain, memiliki jumlah positif benar yang sangat kecil (4), terutama dibandingkan dengan jumlah negatif palsu dan positif palsu—ada lebih banyak positif palsu daripada positif sejati! Prediksi yang dibuat oleh pohon keputusan jauh lebih masuk akal daripada prediksi dummy, meskipun akurasinya hampir sama. Akhirnya, kita dapat melihat bahwa regresi logistik bekerja lebih baik daripada `pred_tree` dalam semua aspek: ia memiliki lebih banyak positif dan negatif sejati sementara memiliki lebih sedikit positif palsu dan negatif palsu. Dari perbandingan ini, jelas bahwa hanya pohon keputusan dan regresi logistik yang memberikan hasil yang masuk akal, dan bahwa regresi logistik bekerja lebih baik daripada pohon pada semua akun. Namun, memeriksa matriks kebingungan penuh agak rumit, dan sementara kami memperoleh banyak wawasan dari melihat semua aspek matriks, prosesnya sangat manual dan kualitatif. Ada beberapa cara untuk meringkas informasi dalam matriks konfusi, yang akan kita bahas selanjutnya.

**Kaitannya dengan akurasi.**

Kita telah melihat satu cara untuk meringkas hasil dalam matriks kebingungan—dengan menghitung akurasi, yang dapat dinyatakan sebagai:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Dengan kata lain, akurasi adalah jumlah prediksi yang benar (TP dan TN) dibagi dengan jumlah semua sampel (semua entri matriks kebingungan dijumlahkan).

Presisi, recall, dan f-score. Ada beberapa cara lain untuk meringkas matriks konfusi, dengan yang paling umum adalah presisi dan ingatan. Presisi mengukur berapa banyak sampel yang diprediksi positif ternyata positif:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Presisi digunakan sebagai metrik kinerja ketika tujuannya adalah untuk membatasi jumlah positif palsu. Sebagai contoh, bayangkan sebuah model untuk memprediksi apakah obat baru akan efektif dalam mengobati penyakit dalam uji klinis. Uji klinis terkenal mahal,

dan perusahaan farmasi hanya ingin menjalankan eksperimen jika sangat yakin bahwa obat itu benar-benar berfungsi. Oleh karena itu, penting agar model tersebut tidak menghasilkan banyak positif palsu—dengan kata lain, model tersebut memiliki presisi yang tinggi. Presisi juga dikenal sebagai nilai prediksi positif (PPV).

Ingat, di sisi lain, mengukur berapa banyak sampel positif yang ditangkap oleh prediksi positif:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

Recall digunakan sebagai metrik kinerja ketika kita perlu mengidentifikasi semua sampel positif; yaitu, ketika penting untuk menghindari negatif palsu. Contoh diagnosis kanker dari awal bab ini adalah contoh yang baik untuk ini: penting untuk menemukan semua orang yang sakit, mungkin termasuk pasien yang sehat dalam prediksi. Nama lain untuk recall adalah sensitivitas, hit rate, atau true positive rate (TPR).

Ada trade-off antara mengoptimalkan penarikan dan mengoptimalkan presisi. Anda dapat memperoleh ingatan yang sempurna jika Anda memperkirakan semua sampel termasuk dalam kelas positif—tidak akan ada negatif palsu, dan juga tidak ada negatif sejati. Namun, memprediksi semua sampel sebagai positif akan menghasilkan banyak positif palsu, dan oleh karena itu presisi akan sangat rendah. Di sisi lain, jika Anda menemukan model yang memprediksi hanya satu titik data yang paling pasti sebagai positif dan sisanya sebagai negatif, maka presisi akan sempurna (dengan asumsi titik data ini sebenarnya positif), tetapi recall akan menjadi sangat buruk.



Presisi dan recall hanyalah dua dari banyak ukuran klasifikasi yang diturunkan dari TP, FP, TN, dan FN. Anda dapat menemukan ringkasan yang bagus dari semua tindakan di Wikipedia. Dalam komunitas pembelajaran mesin, presisi dan ingatan bisa dibilang merupakan ukuran yang paling umum digunakan untuk klasifikasi biner, tetapi komunitas lain mungkin menggunakan metrik terkait lainnya.

Jadi, sementara presisi dan daya ingat adalah ukuran yang sangat penting, melihat hanya satu dari mereka tidak akan memberi Anda gambaran lengkap. Salah satu cara untuk meringkasnya adalah f-score atau f-measure, yaitu dengan rata-rata harmonik presisi dan recall:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Varian khusus ini juga dikenal sebagai f1-score. Karena memperhitungkan presisi dan ingatan, ini bisa menjadi ukuran yang lebih baik daripada akurasi pada kumpulan data klasifikasi biner yang tidak seimbang. Mari kita jalankan pada prediksi untuk kumpulan data “sembilan vs. istirahat” yang telah kita hitung sebelumnya. Di sini, kita akan mengasumsikan bahwa kelas “sembilan” adalah kelas positif (diberi label True sedangkan sisanya diberi label False), sehingga kelas positif adalah kelas minoritas:

**In[45]:**

```
from sklearn.metrics import f1_score
print("f1 score most frequent: {:.2f}".format(
    f1_score(y_test, pred_most_frequent)))
print("f1 score dummy: {:.2f}".format(f1_score(y_test, pred_dummy)))
print("f1 score tree: {:.2f}".format(f1_score(y_test, pred_tree)))
print("f1 score logistic regression: {:.2f}".format(
    f1_score(y_test, pred_logreg)))
```

**Out[45]:**

```
f1 score most frequent: 0.00
f1 score dummy: 0.10
f1 score tree: 0.55
f1 score logistic regression: 0.89
```

Kita dapat mencatat dua hal di sini. Pertama, kita mendapatkan pesan kesalahan untuk prediksi paling\_sering, karena tidak ada prediksi dari kelas positif (yang membuat penyebut pada nilai f menjadi nol). Selain itu, kita dapat melihat perbedaan yang cukup kuat antara prediksi dummy dan prediksi pohon, yang tidak jelas ketika melihat akurasi saja. Menggunakan f-score untuk evaluasi, kami merangkum kinerja prediktif lagi dalam satu nomor. Namun, f-score tampaknya menangkap intuisi kita tentang apa yang membuat model yang bagus jauh lebih baik daripada akurasi. Kelemahan dari f-score, bagaimanapun, adalah lebih sulit untuk ditafsirkan dan dijelaskan daripada akurasi.

Jika kita menginginkan ringkasan presisi, recall, dan f1-score yang lebih komprehensif, kita dapat menggunakan fungsi kemudahan classification\_report untuk menghitung ketiganya sekaligus, dan mencetaknya dalam format yang bagus:

**In[46]:**

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred_most_frequent,
                            target_names=["not nine", "nine"]))
```

**Out[46]:**

	precision	recall	f1-score	support
not nine	0.90	1.00	0.94	403
nine	0.00	0.00	0.00	47
avg / total	0.80	0.90	0.85	450

Fungsi klasifikasi\_laporan menghasilkan satu baris per kelas (di sini, Benar dan Salah) dan melaporkan presisi, ingatan, dan skor-f dengan kelas ini sebagai kelas positif. Sebelumnya, kita asumsikan kelas minoritas “sembilan” adalah kelas positif. Jika kita mengubah kelas positif menjadi “bukan sembilan”, kita dapat melihat dari keluaran klasifikasi\_laporan bahwa kita memperoleh skor-f 0,94 dengan model paling\_sering. Selanjutnya, untuk kelas “bukan sembilan” kami memiliki penarikan 1, karena kami mengklasifikasikan semua sampel sebagai “bukan sembilan.” Kolom terakhir di sebelah f-score memberikan dukungan dari setiap kelas, yang berarti jumlah sampel di kelas ini sesuai dengan kebenaran dasar.

Baris terakhir dalam laporan klasifikasi menunjukkan rata-rata tertimbang (berdasarkan jumlah sampel dalam kelas) dari angka untuk setiap kelas. Berikut adalah dua laporan lagi, satu untuk pengklasifikasi dummy dan satu untuk regresi logistik:

**In[47]:**

```
print(classification_report(y_test, pred_dummy,
                            target_names=["not nine", "nine"]))
```

**Out[47]:**

	precision	recall	f1-score	support
not nine	0.90	0.92	0.91	403
nine	0.11	0.09	0.10	47
avg / total	0.81	0.83	0.82	450

**In[48]:**

```
print(classification_report(y_test, pred_logreg,
                            target_names=["not nine", "nine"]))
```

**Out[48]:**

	precision	recall	f1-score	support
not nine	0.98	1.00	0.99	403
nine	0.95	0.83	0.89	47
avg / total	0.98	0.98	0.98	450

Seperti yang mungkin Anda perhatikan ketika melihat laporan, perbedaan antara model dummy dan model yang sangat bagus tidak begitu jelas lagi. Memilih kelas mana yang dinyatakan sebagai kelas positif memiliki dampak besar pada metrik. Sementara nilai-f untuk klasifikasi dummy adalah 0,13 (vs. 0,89 untuk regresi logistik) pada kelas “sembilan”, untuk kelas “bukan sembilan” adalah 0,90 vs. 0,99, yang keduanya tampak seperti hasil yang masuk akal. Melihat semua angka bersama-sama memberikan gambaran yang cukup akurat, dan kita dapat dengan jelas melihat keunggulan model regresi logistik.

### Mempertimbangkan ketidakpastian

Matriks kebingungan dan laporan klasifikasi memberikan analisis yang sangat rinci dari serangkaian prediksi tertentu. Namun, prediksi itu sendiri sudah membuang banyak informasi yang terkandung dalam model. Seperti yang telah kita diskusikan di Bab 2, kebanyakan pengklasifikasi menyediakan fungsi-keputusan atau metode prediksi\_proba untuk menilai derajat kepastian tentang prediksi. Membuat prediksi dapat dilihat sebagai ambang keluaran dari fungsi\_keputusan atau prediksi\_proba pada titik tetap tertentu—dalam klasifikasi biner kami menggunakan 0 untuk fungsi keputusan dan 0,5 untuk prediksi\_proba.

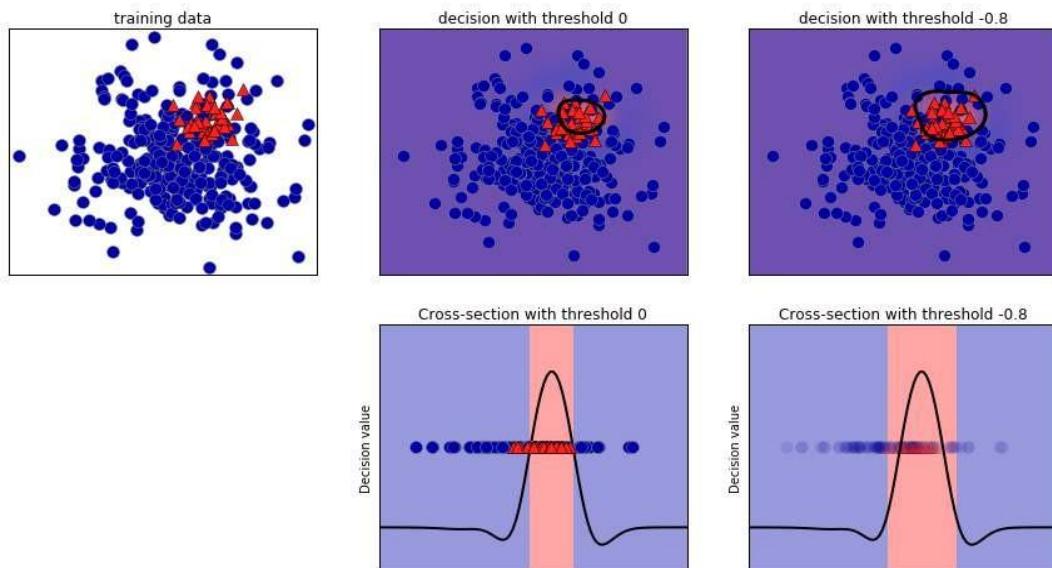
Berikut ini adalah contoh tugas klasifikasi biner tidak seimbang, dengan 400 poin di kelas negatif diklasifikasikan terhadap 50 poin di kelas positif. Data pelatihan ditunjukkan di sebelah kiri pada Gambar 5.12. Kami melatih model SVM kernel pada data ini, dan plot di sebelah kanan data pelatihan menggambarkan nilai fungsi keputusan sebagai peta panas. Anda dapat melihat lingkaran hitam di plot di tengah atas, yang menunjukkan ambang batas dari fungsi\_keputusan menjadi persis nol. Poin di dalam lingkaran ini akan diklasifikasikan sebagai kelas positif, dan poin di luar sebagai kelas negatif:

In[49]:

```
from mglearn.datasets import make_blobs
X, y = make_blobs(n_samples=(400, 50), centers=2, cluster_std=[7.0, 2],
                  random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
svc = SVC(gamma=.05).fit(X_train, y_train)
```

In[50]:

```
mglearn.plots.plot_decision_threshold()
```



**Gambar 5.12** Heatmap fungsi keputusan dan dampak perubahan ambang batas keputusan

Kita dapat menggunakan fungsi klasifikasi\_laporan untuk mengevaluasi presisi dan penarikan kembali untuk kedua kelas:

In[51]:

```
print(classification_report(y_test, svc.predict(X_test)))
```

Out[51]:

	precision	recall	f1-score	support
0	0.97	0.89	0.93	104
1	0.35	0.67	0.46	9
avg / total	0.92	0.88	0.89	113

Untuk kelas 1, kami mendapatkan penarikan yang cukup kecil, dan presisinya beragam. Karena kelas 0 jauh lebih besar, pengklasifikasi berfokus untuk mendapatkan kelas 0, bukan kelas yang lebih kecil 1.

Mari kita asumsikan dalam aplikasi kita lebih penting untuk memiliki daya ingat yang tinggi untuk kelas 1, seperti pada contoh skrining kanker sebelumnya. Ini berarti kita bersedia mengambil risiko lebih banyak positif palsu (kelas palsu 1) dengan imbalan lebih banyak positif benar (yang akan meningkatkan penarikan). Prediksi yang dihasilkan oleh svc.predict benar-benar tidak memenuhi persyaratan ini, tetapi kita dapat menyesuaikan prediksi untuk fokus pada recall kelas 1 yang lebih tinggi dengan mengubah ambang batas keputusan dari 0. Secara

default, poin dengan nilai `decision_function` lebih besar dari 0 akan diklasifikasikan sebagai kelas 1. Kami ingin lebih banyak poin untuk diklasifikasikan sebagai kelas 1, jadi kami perlu mengurangi ambang batas:

**In[52]:**

```
y_pred_lower_threshold = svc.decision_function(X_test) > - .8
```

Mari kita lihat laporan klasifikasi untuk prediksi ini:

**In[53]:**

```
print(classification_report(y_test, y_pred_lower_threshold))
```

**Out[53]:**

	precision	recall	f1-score	support
0	1.00	0.82	0.90	104
1	0.32	1.00	0.49	9
avg / total	0.95	0.83	0.87	113

Seperti yang diharapkan, ingatan kelas 1 naik, dan presisi turun. Kami sekarang mengklasifikasikan wilayah ruang yang lebih besar sebagai kelas 1, seperti yang diilustrasikan di panel kanan atas Gambar 5.12. Jika Anda lebih menghargai presisi daripada mengingat atau sebaliknya, atau data Anda sangat tidak seimbang, mengubah ambang batas keputusan adalah cara termudah untuk mendapatkan hasil yang lebih baik. Karena fungsi\_keputusan dapat memiliki rentang arbitrer, sulit untuk memberikan aturan praktis tentang cara memilih ambang batas.



Jika Anda menetapkan ambang batas, Anda harus berhati-hati untuk tidak melakukannya menggunakan set pengujian. Seperti parameter lainnya, menetapkan ambang keputusan pada set pengujian kemungkinan akan menghasilkan hasil yang terlalu optimis. Gunakan set validasi atau validasi silang sebagai gantinya.

Memilih ambang batas untuk model yang mengimplementasikan metode `predict_proba` bisa lebih mudah, karena output dari `predict_proba` berada pada skala 0 hingga 1 yang tetap, dan probabilitas model. Secara default, ambang batas 0,5 berarti bahwa jika model lebih dari 50% "yakin" bahwa suatu titik adalah kelas positif, maka akan diklasifikasikan seperti itu. Meningkatkan ambang berarti bahwa model perlu lebih percaya diri untuk membuat keputusan positif (dan kurang percaya diri untuk membuat keputusan negatif). Meskipun bekerja dengan probabilitas mungkin lebih intuitif daripada bekerja dengan ambang batas yang sewenang-wenang, tidak semua model menyediakan model ketidakpastian yang realistik (DecisionTree yang tumbuh hingga kedalaman penuhnya selalu 100% yakin akan keputusannya, meskipun mungkin sering salah). Hal ini berkaitan dengan konsep kalibrasi: model yang dikalibrasi adalah model yang memberikan ukuran ketidakpastian yang akurat. Membahas kalibrasi secara mendetail berada di luar cakupan buku ini, tetapi Anda dapat menemukan detail lebih lanjut dalam makalah "Memprediksi Probabilitas Baik dengan Pembelajaran yang Dibimbing" oleh Alexandru Niculescu-Mizil dan Rich Caruana.

## Kurva presisi-recall dan kurva ROC

Seperti yang baru saja kita diskusikan, mengubah ambang batas yang digunakan untuk membuat keputusan klasifikasi dalam model adalah cara untuk menyesuaikan trade-off presisi dan recall untuk pengklasifikasi tertentu. Mungkin Anda ingin melewatkannya kurang dari 10% sampel positif, artinya penarikan kembali yang diinginkan sebesar 90%. Keputusan ini tergantung pada aplikasinya, dan harus didorong oleh tujuan bisnis. Setelah tujuan tertentu ditetapkan—misalnya, ingatan atau nilai presisi tertentu untuk suatu kelas—ambang batas dapat ditetapkan dengan tepat. Selalu mungkin untuk menetapkan ambang batas untuk memenuhi target tertentu, seperti penarikan 90%. Bagian tersulitnya adalah mengembangkan model yang masih memiliki presisi yang masuk akal dengan ambang batas ini—if Anda mengklasifikasikan semuanya sebagai positif, Anda akan memiliki ingatan 100%, tetapi model Anda tidak akan berguna.

Menetapkan persyaratan pada pengklasifikasi seperti penarikan 90% sering disebut pengaturan titik operasi. Memperbaiki titik operasi sering membantu dalam pengaturan bisnis untuk membuat jaminan kinerja kepada pelanggan atau kelompok lain di dalam organisasi Anda.

Seringkali, ketika mengembangkan model baru, tidak sepenuhnya jelas apa titik operasinya. Untuk alasan ini, dan untuk memahami masalah pemodelan dengan lebih baik, penting untuk melihat semua ambang batas yang mungkin, atau semua kemungkinan pertukaran presisi dan penarikan sekaligus. Hal ini dimungkinkan dengan menggunakan alat yang disebut kurva precision-recall. Anda dapat menemukan fungsi untuk menghitung kurva presisi-recall di modul `sklearn.metrics`. Ini membutuhkan pelabelan kebenaran dasar dan ketidakpastian yang diprediksi, dibuat baik melalui `decision_function` atau `predict_proba`:

**In[54]:**

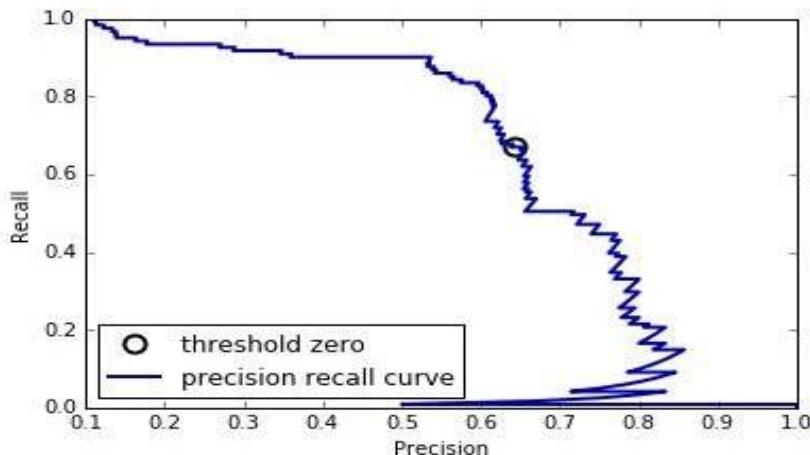
```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(
    y_test, svc.decision_function(X_test))
```

Fungsi `precision_recall_curve` mengembalikan daftar nilai presisi dan recall untuk semua ambang batas yang mungkin (semua nilai yang muncul dalam fungsi keputusan) dalam urutan yang diurutkan, sehingga kita dapat memplot kurva, seperti yang terlihat pada Gambar 5.13:

**In[55]:**

```
# Use more data points for a smoother curve
X, y = make_blobs(n_samples=(4000, 500), centers=2, cluster_std=[7.0, 2],
                  random_state=22)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
svc = SVC(gamma=.05).fit(X_train, y_train)
precision, recall, thresholds = precision_recall_curve(
    y_test, svc.decision_function(X_test))
# find threshold closest to zero
close_zero = np.argmin(np.abs(thresholds))
plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10,
          label="threshold zero", fillstyle="none", c='k', mew=2)

plt.plot(precision, recall, label="precision recall curve")
plt.xlabel("Precision")
plt.ylabel("Recall")
```



**Gambar 5.13** Kurva penarikan presisi untuk SVC ( $\gamma=0,05$ )

Setiap titik di sepanjang kurva pada Gambar 5.13 sesuai dengan ambang batas yang mungkin dari fungsi\_keputusan. Kita dapat melihat, misalnya, bahwa kita dapat mencapai penarikan kembali 0,4 pada presisi sekitar 0,75. Lingkaran hitam menandai titik yang sesuai dengan ambang 0, ambang default untuk fungsi\_keputusan. Poin ini adalah trade-off yang dipilih saat memanggil metode predksi.

Semakin dekat kurva ke sudut kanan atas, semakin baik pengklasifikasinya. Titik di kanan atas berarti presisi tinggi dan daya ingat tinggi untuk ambang batas yang sama. Kurva dimulai di sudut kiri atas, sesuai dengan ambang batas yang sangat rendah, mengklasifikasikan semuanya sebagai kelas positif. Menaikkan ambang batas menggerakkan kurva menuju presisi yang lebih tinggi, tetapi juga menurunkan daya ingat. Menaikkan ambang batas lebih dan lebih, kita sampai pada situasi di mana sebagian besar poin yang diklasifikasikan sebagai positif adalah benar-benar positif, yang mengarah ke presisi yang sangat tinggi tetapi daya ingat yang lebih rendah. Semakin model terus mengingat tinggi saat presisi meningkat, semakin baik.

Melihat kurva khusus ini sedikit lebih, kita dapat melihat bahwa dengan model ini dimungkinkan untuk mendapatkan presisi hingga sekitar 0,5 dengan daya ingat yang sangat tinggi. Jika kita ingin presisi yang jauh lebih tinggi, kita harus mengorbankan banyak recall. Dengan kata lain, di sebelah kiri kurvanya relatif datar, artinya recall tidak turun banyak ketika kita membutuhkan peningkatan presisi. Untuk presisi yang lebih besar dari 0,5, setiap perolehan presisi membutuhkan banyak penarikan.

Pengklasifikasi yang berbeda dapat bekerja dengan baik di bagian kurva yang berbeda—yaitu, pada titik operasi yang berbeda. Mari kita bandingkan SVM yang kita latih dengan hutan acak yang dilatih pada dataset yang sama. RandomForestClassifier tidak memiliki `decision_function`, hanya `predict_proba`. Fungsi `precision_recall_curve` mengharapkan sebagai argumen kedua ukuran kepastian untuk kelas positif (kelas 1), jadi kami melewatkannya `probabilitas sampel menjadi kelas 1`—yaitu, `rf.predict_proba(X_test)[:, 1]`. Ambang default untuk `predict_proba` dalam klasifikasi biner adalah 0,5, jadi ini adalah titik yang kita tandai pada kurva (lihat Gambar 5.14):

In[56]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0, max_features=2)
rf.fit(X_train, y_train)

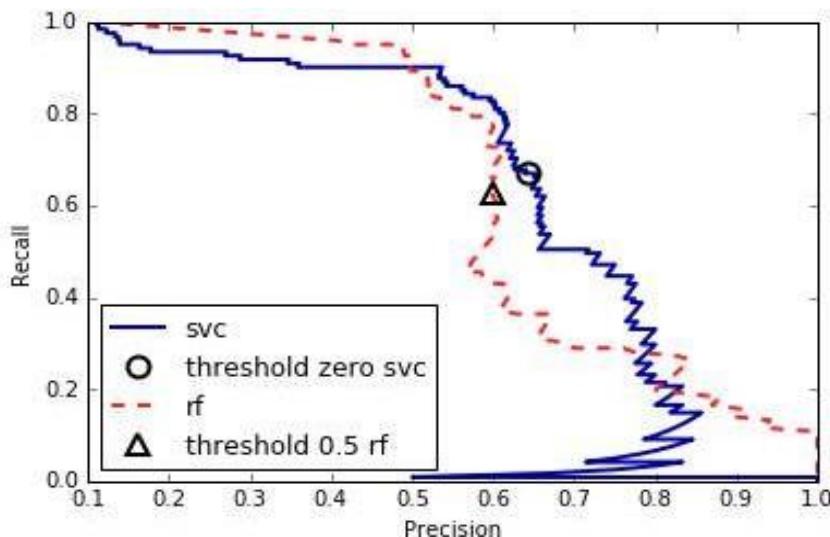
# RandomForestClassifier has predict_proba, but not decision_function
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(
    y_test, rf.predict_proba(X_test)[:, 1])

plt.plot(precision, recall, label="svc")

plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10,
         label="threshold zero svc", fillstyle="none", c='k', mew=2)

plt.plot(precision_rf, recall_rf, label="rf")

close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))
plt.plot(precision_rf[close_default_rf], recall_rf[close_default_rf], '^', c='k',
         markersize=10, label="threshold 0.5 rf", fillstyle="none", mew=2)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.legend(loc="best")
```



Gambar 5.14 Membandingkan kurva penarikan presisi SVM dan hutan acak

Dari plot perbandingan, kita dapat melihat bahwa hutan acak berkinerja lebih baik pada kondisi ekstrem, untuk penarikan yang sangat tinggi atau persyaratan presisi yang sangat tinggi. Sekitar tengah (kurang lebih presisi = 0,7), kinerja SVM lebih baik. Jika kita hanya melihat f1-score untuk membandingkan kinerja secara keseluruhan, kita akan melewatkannya. F1-score hanya menangkap satu titik pada kurva presisi-recall, yang diberikan oleh ambang default:

In[57]:

```
print("f1_score of random forest: {:.3f}".format(
    f1_score(y_test, rf.predict(X_test))))
print("f1_score of svc: {:.3f}".format(f1_score(y_test, svc.predict(X_test))))
```

Out[57]:

```
f1_score of random forest: 0.610
f1_score of svc: 0.656
```

Membandingkan dua kurva presisi-recall memberikan banyak wawasan rinci, tetapi merupakan proses yang cukup manual. Untuk perbandingan model otomatis, kita mungkin ingin meringkas informasi yang terkandung dalam kurva, tanpa membatasi diri pada ambang atau titik operasi tertentu. Salah satu cara khusus untuk meringkas kurva presisi-ingat adalah dengan menghitung integral atau area di bawah kurva kurva presisi-ingat, juga dikenal sebagai presisi rata-rata.<sup>4</sup> Anda dapat menggunakan fungsi `average_precision_score` untuk menghitung presisi rata-rata. Karena kita perlu menghitung kurva ROC dan mempertimbangkan beberapa ambang batas, hasil dari `decision_function` atau `predict_proba` perlu diteruskan ke `average_precision_score`, bukan hasil dari `predict`:

**In[58]:**

```
from sklearn.metrics import average_precision_score
ap_rf = average_precision_score(y_test, rf.predict_proba(X_test)[:, 1])
ap_svc = average_precision_score(y_test, svc.decision_function(X_test))
print("Average precision of random forest: {:.3f}".format(ap_rf))
print("Average precision of svc: {:.3f}".format(ap_svc))
```

**Out[58]:**

```
Average precision of random forest: 0.666
Average precision of svc: 0.663
```

Saat merata-ratakan semua ambang batas yang mungkin, kami melihat bahwa hutan acak dan SVC berkinerja sama baiknya, dengan hutan acak sedikit di depan. Hal ini cukup berbeda dengan hasil yang kami peroleh dari `f1_score` tadi. Karena presisi rata-rata adalah area di bawah kurva yang berkisar dari 0 hingga 1, presisi rata-rata selalu mengembalikan nilai antara 0 (terburuk) dan 1 (terbaik). Ketepatan rata-rata pengklasifikasi yang menetapkan fungsi\_keputusan secara acak adalah fraksi sampel positif dalam kumpulan data.

### Karakteristik operasi penerima (ROC) dan AUC

Ada alat lain yang biasa digunakan untuk menganalisis perilaku pengklasifikasi pada ambang batas yang berbeda: kurva karakteristik operasi penerima, atau singkatnya kurva ROC. Mirip dengan kurva presisi-recall, kurva ROC mempertimbangkan semua ambang batas yang mungkin untuk pengklasifikasi tertentu, tetapi alih-alih melaporkan presisi dan recall, kurva ini menunjukkan tingkat positif palsu (FPR) terhadap tingkat positif benar (TPR). Ingat bahwa tingkat positif benar hanyalah nama lain untuk mengingat, sedangkan tingkat positif palsu adalah fraksi dari positif palsu dari semua sampel negatif:

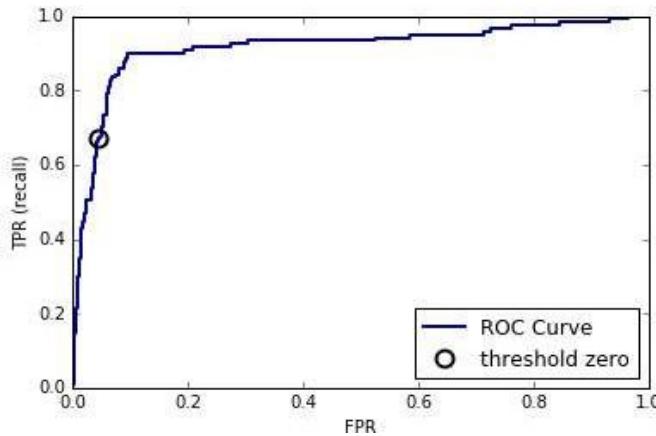
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

Kurva ROC dapat dihitung menggunakan fungsi `roc_curve` (lihat Gambar 5.15):

In[59]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))

plt.plot(fpr, tpr, label="ROC Curve")
plt.xlabel("FPR")
plt.ylabel("TPR (recall)")
# find threshold closest to zero
close_zero = np.argmin(np.abs(thresholds))
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
         label="threshold zero", fillstyle="none", c='k', mew=2)
plt.legend(loc=4)
```



Gambar 5.15 Kurva ROC untuk SVM

Untuk kurva ROC, kurva ideal dekat dengan kiri atas: Anda menginginkan classifier yang menghasilkan recall tinggi sambil mempertahankan tingkat false positive yang rendah. Dibandingkan dengan ambang batas default 0, kurva menunjukkan bahwa kita dapat mencapai penarikan yang jauh lebih tinggi (sekitar 0,9) sementara hanya meningkatkan FPR sedikit. Titik yang paling dekat dengan kiri atas mungkin merupakan titik pengoperasian yang lebih baik daripada titik yang dipilih secara default. Sekali lagi, ketahuilah bahwa memilih ambang batas tidak boleh dilakukan pada set pengujian, tetapi pada set validasi yang terpisah.

Anda dapat menemukan perbandingan hutan acak dan SVM menggunakan kurva ROC pada Gambar 5.16:

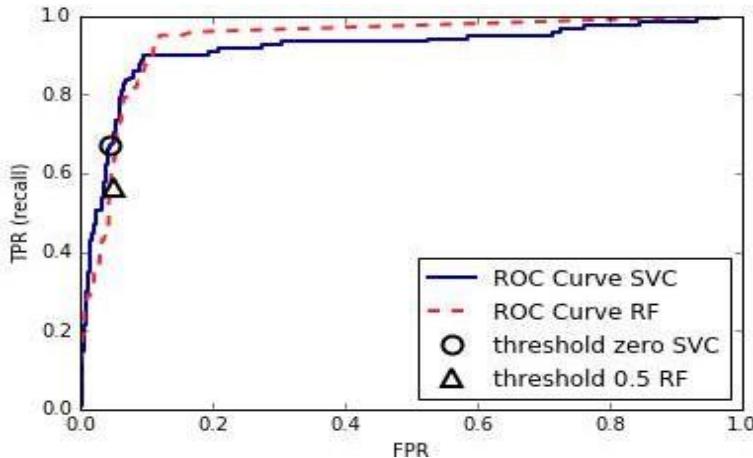
In[60]:

```
from sklearn.metrics import roc_curve
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, rf.predict_proba(X_test)[:, 1])

plt.plot(fpr, tpr, label="ROC Curve SVC")
plt.plot(fpr_rf, tpr_rf, label="ROC Curve RF")

plt.xlabel("FPR")
plt.ylabel("TPR (recall)")
plt.plot(fpr[close_zero], tpr[close_zero], 'o', markersize=10,
         label="threshold zero SVC", fillstyle="none", c='k', mew=2)
close_default_rf = np.argmin(np.abs(thresholds_rf - 0.5))
plt.plot(fpr_rf[close_default_rf], tpr_rf[close_default_rf], '^', markersize=10,
         label="threshold 0.5 RF", fillstyle="none", c='k', mew=2)

plt.legend(loc=4)
```



**Gambar 5.16** Membandingkan kurva ROC untuk SVM dan hutan acak

Sedangkan untuk kurva presisi-recall, kita sering ingin meringkas kurva ROC menggunakan satu angka, area di bawah kurva (ini biasanya hanya disebut sebagai AUC, dan dapat dipahami bahwa kurva yang dimaksud adalah kurva ROC.). Kita dapat menghitung luas di bawah kurva ROC menggunakan fungsi `roc_auc_score`:

**In[61]:**

```
from sklearn.metrics import roc_auc_score
rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
svc_auc = roc_auc_score(y_test, svc.decision_function(X_test))
print("AUC for Random Forest: {:.3f}".format(rf_auc))
print("AUC for SVC: {:.3f}".format(svc_auc))
```

**Out[61]:**

```
AUC for Random Forest: 0.937
AUC for SVC: 0.916
```

Membandingkan hutan acak dan SVM menggunakan skor AUC, kami menemukan bahwa hutan acak berkinerja sedikit lebih baik daripada SVM. Ingatlah bahwa karena presisi rata-rata adalah area di bawah kurva yang berkisar dari 0 hingga 1, presisi rata-rata selalu mengembalikan nilai antara 0 (terburuk) dan 1 (terbaik). Memprediksi secara acak selalu menghasilkan AUC 0,5, tidak peduli seberapa tidak seimbangnya kelas dalam kumpulan data. Ini menjadikan AUC metrik yang jauh lebih baik untuk masalah klasifikasi yang tidak seimbang daripada akurasi. AUC dapat diartikan sebagai evaluasi peringkat sampel positif. Ini setara dengan probabilitas bahwa titik yang diambil secara acak dari kelas positif akan memiliki skor yang lebih tinggi menurut pengklasifikasi daripada titik yang diambil secara acak dari kelas negatif. Jadi, AUC sempurna 1 berarti semua poin positif memiliki skor lebih tinggi daripada semua poin negatif. Untuk masalah klasifikasi dengan kelas yang tidak seimbang, menggunakan AUC untuk pemilihan model seringkali jauh lebih bermakna daripada menggunakan akurasi.

Mari kembali ke masalah yang kita pelajari sebelumnya tentang mengklasifikasikan semua sembilan dalam kumpulan data digit versus semua digit lainnya. Kami akan mengklasifikasikan dataset dengan SVM dengan tiga pengaturan berbeda dari bandwidth kernel, gamma (lihat Gambar 5.17):

**In[62]:**

```

y = digits.target == 9

X_train, X_test, y_train, y_test = train_test_split(
    digits.data, y, random_state=0)

plt.figure()

for gamma in [1, 0.05, 0.01]:
    svc = SVC(gamma=gamma).fit(X_train, y_train)
    accuracy = svc.score(X_test, y_test)
    auc = roc_auc_score(y_test, svc.decision_function(X_test))
    fpr, tpr, _ = roc_curve(y_test, svc.decision_function(X_test))
    print("gamma = {:.2f} accuracy = {:.2f} AUC = {:.2f}".format(
        gamma, accuracy, auc))
    plt.plot(fpr, tpr, label="gamma={:.3f}".format(gamma))
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.xlim(-0.01, 1)
plt.ylim(0, 1.02)
plt.legend(loc="best")

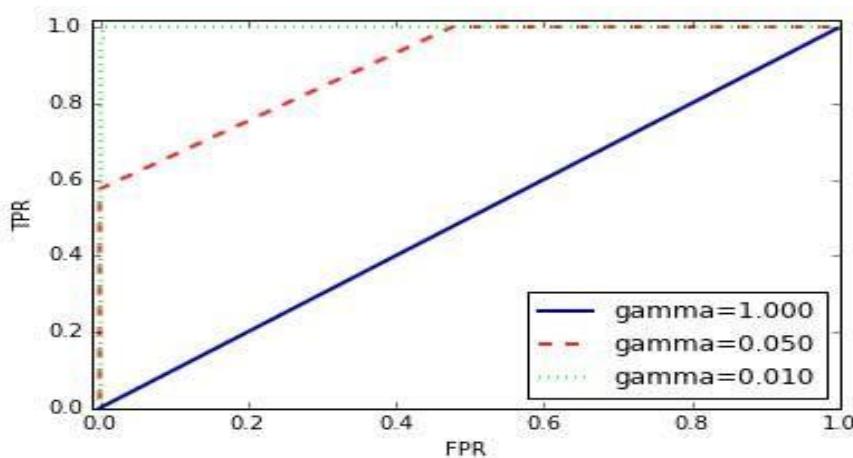
```

**Out[62]:**

```

gamma = 1.00  accuracy = 0.90  AUC = 0.50
gamma = 0.05  accuracy = 0.90  AUC = 0.90
gamma = 0.01  accuracy = 0.90  AUC = 1.00

```



**Gambar 5.17** Membandingkan kurva ROC dari SVM dengan pengaturan gamma yang berbeda

Keakuratan ketiga pengaturan gamma adalah sama, 90%. Ini mungkin sama dengan kinerja kebetulan, atau mungkin tidak. Namun, melihat AUC dan kurva yang sesuai, kami melihat perbedaan yang jelas antara ketiga model tersebut. Dengan  $\text{gamma}=1,0$ , AUC sebenarnya berada pada tingkat kebetulan, yang berarti bahwa output dari fungsi\_keputusan sama baiknya dengan acak. Dengan  $\text{gamma}=0,05$ , kinerja meningkat drastis ke AUC 0,5. Akhirnya, dengan  $\text{gamma}=0,01$ , kita mendapatkan AUC sempurna 1,0. Itu berarti bahwa semua poin positif diberi peringkat lebih tinggi dari semua poin negatif menurut fungsi keputusan. Dengan kata lain, dengan ambang batas yang tepat, model ini dapat mengklasifikasikan data dengan sempurna! Mengetahui hal ini, kita dapat menyesuaikan

ambang batas pada model ini dan mendapatkan prediksi yang bagus. Jika kami hanya menggunakan akurasi, kami tidak akan pernah menemukan ini.

Untuk alasan ini, kami sangat merekomendasikan penggunaan AUC saat mengevaluasi model pada data yang tidak seimbang. Perlu diingat bahwa AUC tidak menggunakan ambang default, jadi menyesuaikan ambang keputusan mungkin diperlukan untuk mendapatkan hasil klasifikasi yang berguna dari model dengan AUC tinggi.

## 5.8 METRIK UNTUK KLASIFIKASI MULTIKLAS

Sekarang kita telah membahas evaluasi tugas klasifikasi biner secara mendalam, mari beralih ke metrik untuk mengevaluasi klasifikasi multikelas. Pada dasarnya, semua metrik untuk klasifikasi multikelas diturunkan dari metrik klasifikasi biner, tetapi dirata-ratakan untuk semua kelas. Akurasi untuk klasifikasi multikelas sekali lagi didefinisikan sebagai pecahan dari contoh yang diklasifikasikan dengan benar. Dan lagi, ketika kelas tidak seimbang, akurasi bukanlah ukuran evaluasi yang bagus. Bayangkan masalah klasifikasi tiga kelas dengan 85% poin milik kelas A, 10% milik kelas B, dan 5% milik kelas C. Apa artinya menjadi 85% akurat pada kumpulan data ini? Secara umum, hasil klasifikasi multiklasifikasi lebih sulit dipahami daripada hasil klasifikasi biner. Terlepas dari akurasi, alat umum adalah matriks kebingungan dan laporan klasifikasi yang kita lihat dalam kasus biner di bagian sebelumnya. Mari kita terapkan dua metode evaluasi terperinci ini pada tugas mengklasifikasikan 10 digit tulisan tangan yang berbeda dalam kumpulan data digit:

**In[63]:**

```
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

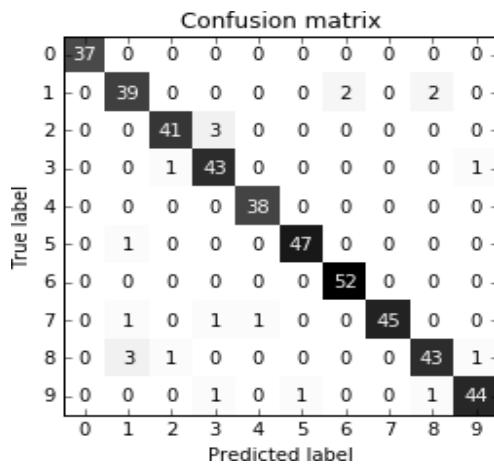
**Out[63]:**

```
Accuracy: 0.953
Confusion matrix:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

Model ini memiliki akurasi 95,3%, yang sudah memberi tahu kami bahwa kami melakukannya dengan cukup baik. Matriks kebingungan memberi kita beberapa detail lebih lanjut. Adapun kasus biner, setiap baris sesuai dengan label yang benar, dan setiap kolom sesuai dengan label yang diprediksi. Anda dapat menemukan plot yang lebih menarik secara visual pada Gambar 5.18:

In[64]:

```
scores_image = mglearn.tools.heatmap(
    confusion_matrix(y_test, pred), xlabel='Predicted label',
    ylabel='True label', xticklabels=digits.target_names,
    yticklabels=digits.target_names, cmap=plt.cm.gray_r, fmt="%d")
plt.title("Confusion matrix")
plt.gca().invert_yaxis()
```



Gambar 5.18 Matriks kebingungan untuk tugas klasifikasi 10 digit

Untuk kelas pertama, angka 0, ada 37 sampel di kelas, dan semua sampel ini diklasifikasikan sebagai kelas 0 (tidak ada negatif palsu untuk kelas 0). Kita dapat melihat bahwa karena semua entri lain di baris pertama matriks konfusi adalah 0. Kita juga dapat melihat bahwa tidak ada angka lain yang salah diklasifikasikan sebagai 0, karena semua entri lain di kolom pertama matriks konfusi adalah 0 (ada tidak ada positif palsu untuk kelas 0). Namun, beberapa digit membingungkan dengan yang lain—misalnya, digit 2 (baris ketiga), tiga di antaranya diklasifikasikan sebagai digit 3 (kolom keempat). Ada juga satu digit 3 yang diklasifikasikan sebagai 2 (kolom ketiga, baris keempat) dan satu digit 8 yang diklasifikasikan sebagai 2 (kolom ketiga, baris keempat).

Dengan fungsi `classification_report`, kita dapat menghitung presisi, recall, dan f-score untuk setiap kelas:

In[65]:

```
print(classification_report(y_test, pred))
```

Out[65]:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	37
1	0.89	0.91	0.90	43
2	0.95	0.93	0.94	44
3	0.90	0.96	0.92	45
4	0.97	1.00	0.99	38
5	0.98	0.98	0.98	48
6	0.96	1.00	0.98	52
7	1.00	0.94	0.97	48
8	0.93	0.90	0.91	48
9	0.96	0.94	0.95	47
avg / total	0.95	0.95	0.95	450

Tidak mengherankan, presisi dan daya ingat adalah 1 sempurna untuk kelas 0, karena tidak ada kebingungan dengan kelas ini. Untuk kelas 7, di sisi lain, presisi adalah 1 karena tidak ada kelas lain yang salah diklasifikasikan sebagai 7, sedangkan untuk kelas 6, tidak ada negatif palsu, sehingga recall adalah 1. Kita juga dapat melihat bahwa model memiliki kesulitan khusus dengan kelas 8 dan 3.

Metrik yang paling umum digunakan untuk kumpulan data yang tidak seimbang dalam pengaturan multikelas adalah versi multikelas dari f-score. Ide dibalik multiclass f-score adalah untuk menghitung satu biner f-score per kelas, dengan kelas itu menjadi kelas positif dan kelas lain membentuk kelas negatif. Kemudian, nilai f per kelas ini dirata-ratakan menggunakan salah satu strategi berikut:

- Rata-rata "makro" menghitung skor-f per kelas yang tidak berbobot. Ini memberikan bobot yang sama untuk semua kelas, tidak peduli berapa ukurannya.
- Rata-rata "berbobot" menghitung rata-rata nilai-f per kelas, yang dibobot dengan dukungannya. Inilah yang dilaporkan dalam laporan klasifikasi.
- Rata-rata "mikro" menghitung jumlah total positif palsu, negatif palsu, dan positif benar di semua kelas, dan kemudian menghitung presisi, ingatan, dan skor-f menggunakan penghitungan ini.

Jika Anda sangat peduli dengan setiap sampel, disarankan untuk menggunakan skor f1 rata-rata "mikro"; jika Anda sangat peduli dengan setiap kelas, disarankan untuk menggunakan skor f1 rata-rata "makro":

**In[66]:**

```
print("Micro average f1 score: {:.3f}".format
      (f1_score(y_test, pred, average="micro")))
print("Macro average f1 score: {:.3f}".format
      (f1_score(y_test, pred, average="macro")))
```

**Out[66]:**

```
Micro average f1 score: 0.953
Macro average f1 score: 0.954
```

## 5.9 METRIK REGRESI

Evaluasi untuk regresi dapat dilakukan dengan detail yang sama seperti yang kami lakukan untuk klasifikasi—misalnya, dengan menganalisis prediksi yang berlebihan terhadap target versus prediksi yang kurang dari target. Namun, di sebagian besar aplikasi yang telah kita lihat, menggunakan R2 default yang digunakan dalam metode skor semua regressor sudah cukup. Terkadang keputusan bisnis dibuat berdasarkan kesalahan kuadrat rata-rata atau kesalahan absolut rata-rata, yang mungkin memberikan insentif untuk menyesuaikan model menggunakan metrik ini. Namun, secara umum, kami menemukan R2 sebagai metrik yang lebih intuitif untuk mengevaluasi model regresi.

## 5.10 MENGGUNAKAN METRIK EVALUASI DALAM PEMILIHAN MODEL

Kami telah membahas banyak metode evaluasi secara rinci, dan bagaimana menerapkannya mengingat kebenaran dasar dan sebuah model. Namun, kami sering ingin

menggunakan metrik seperti AUC dalam pemilihan model menggunakan GridSearchCV atau cross\_val\_score. Untungnya scikit-learn menyediakan cara yang sangat sederhana untuk mencapai ini, melalui argumen penilaian yang dapat digunakan di GridSearchCV dan cross\_val\_score. Anda cukup memberikan string yang menjelaskan metrik evaluasi yang ingin Anda gunakan. Katakanlah, misalnya, kami ingin mengevaluasi pengklasifikasi SVM pada tugas "sembilan vs. istirahat" pada kumpulan data digit, menggunakan skor AUC. Mengubah skor dari default (akurasi) ke AUC dapat dilakukan dengan memberikan "roc\_auc" sebagai parameter penilaian:

**In[67]:**

```
# default scoring for classification is accuracy
print("Default scoring: {}".format(
    cross_val_score(SVC(), digits.data, digits.target == 9)))
# providing scoring="accuracy" doesn't change the results
explicit_accuracy = cross_val_score(SVC(), digits.data, digits.target == 9,
                                      scoring="accuracy")
print("Explicit accuracy scoring: {}".format(explicit_accuracy))
roc_auc = cross_val_score(SVC(), digits.data, digits.target == 9,
                           scoring="roc_auc")
print("AUC scoring: {}".format(roc_auc))
```

**Out[67]:**

```
Default scoring: [ 0.9  0.9  0.9]
Explicit accuracy scoring: [ 0.9  0.9  0.9]
AUC scoring: [ 0.994  0.99   0.996]
```

Demikian pula, kami dapat mengubah metrik yang digunakan untuk memilih parameter terbaik di Gridsearchcv:

**In[68]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, digits.target == 9, random_state=0)

# we provide a somewhat bad grid to illustrate the point:
param_grid = {'gamma': [0.0001, 0.01, 0.1, 1, 10]}
# using the default scoring of accuracy:
grid = GridSearchCV(SVC(), param_grid=param_grid)
grid.fit(X_train, y_train)
print("Grid-Search with accuracy")
print("Best parameters:", grid.best_params_)
print("Best cross-validation score (accuracy)): {:.3f}".format(grid.best_score_))
print("Test set AUC: {:.3f}".format(
    roc_auc_score(y_test, grid.decision_function(X_test))))
print("Test set accuracy: {:.3f}".format(grid.score(X_test, y_test)))
```

**Out[68]:**

```
Grid-Search with accuracy
Best parameters: {'gamma': 0.0001}
Best cross-validation score (accuracy)): 0.970
Test set AUC: 0.992
Test set accuracy: 0.973
```

**In[69]:**

```
# using AUC scoring instead:
grid = GridSearchCV(SVC(), param_grid=param_grid, scoring="roc_auc")
grid.fit(X_train, y_train)
print("\nGrid-Search with AUC")
print("Best parameters: ", grid.best_params_)
print("Best cross-validation score (AUC): {:.3f}".format(grid.best_score_))
print("Test set AUC: {:.3f}".format(
    roc_auc_score(y_test, grid.decision_function(X_test))))
print("Test set accuracy: {:.3f}".format(grid.score(X_test, y_test)))
```

**Out[69]:**

```
Grid-Search with AUC
Best parameters: {'gamma': 0.01}
Best cross-validation score (AUC): 0.997
Test set AUC: 1.000
Test set accuracy: 1.000
```

Saat menggunakan akurasi, parameter gamma=0,0001 dipilih, sedangkan gamma=0,01 dipilih saat menggunakan AUC. Akurasi validasi silang konsisten dengan akurasi set tes dalam kedua kasus. Namun, menggunakan AUC menemukan pengaturan parameter yang lebih baik dalam hal AUC dan bahkan dalam hal akurasi.

Nilai yang paling penting untuk parameter penilaian untuk klasifikasi adalah akurasi (default); roc\_auc untuk area di bawah kurva ROC; average\_precision untuk area di bawah kurva precision-recall; f1, f1\_macro, f1\_micro, dan f1\_weighted untuk skor f1 biner dan varian pembobotan yang berbeda. Untuk regresi, nilai yang paling umum digunakan adalah r2 untuk skor R2, mean\_squared\_error untuk mean squared error, dan mean\_absolute\_error untuk mean absolute error. Anda dapat menemukan daftar lengkap argumen yang didukung dalam dokumentasi atau dengan melihat kamus SCORER yang didefinisikan dalam modul metrics.scorer:

**In[70]:**

```
from sklearn.metrics.scorer import SCORERS
print("Available scorers:\n{}".format(sorted(SCORERS.keys())))
```

**Out[70]:**

```
Available scorers:
['accuracy', 'adjusted_rand_score', 'average_precision', 'f1', 'f1_macro',
 'f1_micro', 'f1_samples', 'f1_weighted', 'log_loss', 'mean_absolute_error',
 'mean_squared_error', 'median_absolute_error', 'precision', 'precision_macro',
 'precision_micro', 'precision_samples', 'precision_weighted', 'r2', 'recall',
 'recall_macro', 'recall_micro', 'recall_samples', 'recall_weighted', 'roc_auc']
```

## 5.11 RINGKASAN DAN PANDANGAN

Dalam bab ini kita membahas validasi silang, pencarian grid, dan metrik evaluasi, landasan mengevaluasi dan meningkatkan algoritme pembelajaran mesin. Alat yang dijelaskan dalam bab ini, bersama dengan algoritme yang dijelaskan dalam Bab 2 dan 3, adalah roti dan mentega dari setiap praktisi pembelajaran mesin.

Ada dua poin khusus yang kami buat dalam bab ini yang perlu diulang, karena sering diabaikan oleh praktisi baru. Yang pertama berkaitan dengan validasi silang. Validasi silang

atau penggunaan set pengujian memungkinkan kami mengevaluasi model pembelajaran mesin seperti yang akan dilakukan di masa mendatang. Namun, jika kami menggunakan set uji atau validasi silang untuk memilih model atau memilih parameter model, kami "menggunakan" data uji, dan menggunakan data yang sama untuk mengevaluasi seberapa baik model kami akan dilakukan di masa depan akan menyebabkan terlalu banyak perkiraan optimis. Oleh karena itu, kita perlu menggunakan pemisahan menjadi data pelatihan untuk pembuatan model, data validasi untuk pemilihan model dan parameter, dan data uji untuk evaluasi model. Alih-alih pemisahan sederhana, kita dapat mengganti masing-masing pemisahan ini dengan validasi silang. Bentuk yang paling umum digunakan (seperti yang dijelaskan sebelumnya) adalah pemisahan pelatihan/pengujian untuk evaluasi, dan menggunakan validasi silang pada set pelatihan untuk pemilihan model dan parameter.

Poin kedua berkaitan dengan pentingnya metrik evaluasi atau fungsi penilaian yang digunakan untuk pemilihan model dan evaluasi model. Teori tentang cara membuat keputusan bisnis dari prediksi model pembelajaran mesin agak di luar cakupan buku ini.<sup>7</sup> Namun, jarang terjadi bahwa tujuan akhir tugas pembelajaran mesin adalah membangun model dengan akurasi tinggi. Pastikan bahwa metrik yang Anda pilih untuk dievaluasi dan dipilih modelnya adalah pengganti yang baik untuk tujuan penggunaan model tersebut. Pada kenyataannya, masalah klasifikasi jarang memiliki kelas yang seimbang, dan seringkali positif palsu dan negatif palsu memiliki konsekuensi yang sangat berbeda.

Pastikan Anda memahami apa konsekuensi ini, dan pilih metrik evaluasi yang sesuai. Teknik evaluasi dan pemilihan model yang telah kami jelaskan sejauh ini adalah alat terpenting dalam kotak peralatan ilmuwan data. Pencarian kisi dan validasi silang seperti yang telah kami jelaskan di bab ini hanya dapat diterapkan pada satu model terawasi. Kita telah melihat sebelumnya, bagaimanapun, bahwa banyak model memerlukan pra-pemrosesan, dan bahwa dalam beberapa aplikasi, seperti contoh pengenalan wajah di Bab 3, mengekstraksi representasi data yang berbeda dapat bermanfaat. Pada bab berikutnya, kami akan memperkenalkan kelas Pipeline, yang memungkinkan kami untuk menggunakan pencarian grid dan validasi silang pada rantai algoritma yang kompleks ini.

## BAB 6

### RANTAI DAN PIPA ALGORITMA

Untuk banyak algoritme pembelajaran mesin, representasi khusus dari data yang Anda berikan sangat penting, seperti yang telah kita bahas di Bab 4. Ini dimulai dengan menskalakan data dan menggabungkan fitur dengan tangan dan berlanjut hingga mempelajari fitur menggunakan pembelajaran mesin tanpa pengawasan, seperti yang kita lihat di Bab 3. Akibatnya, sebagian besar aplikasi pembelajaran mesin tidak hanya memerlukan penerapan algoritma tunggal, tetapi juga rantai bersama dari banyak langkah pemrosesan dan model pembelajaran mesin yang berbeda. Dalam bab ini, kita akan membahas cara menggunakan kelas Pipeline untuk menyederhanakan proses membangun rantai transformasi dan model. Secara khusus, kita akan melihat bagaimana kita dapat menggabungkan Pipeline dan GridSearchCV untuk mencari parameter untuk semua langkah pemrosesan sekaligus.

Sebagai contoh pentingnya model rantai, kami memperhatikan bahwa kami dapat sangat meningkatkan kinerja kernel SVM pada kumpulan data kanker dengan menggunakan Min MaxScaler untuk prapemrosesan. Berikut kode untuk memisahkan data, menghitung minimum dan maksimum, menskalakan data, dan melatih SVM:

**In[1]:**

```
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# load and split the data
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

# compute minimum and maximum on the training data
scaler = MinMaxScaler().fit(X_train)
```

**In[2]:**

```
# rescale the training data
X_train_scaled = scaler.transform(X_train)

svm = SVC()
# learn an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)
# scale the test data and score the scaled data
X_test_scaled = scaler.transform(X_test)
print("Test score: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

**Out[2]:**

Test score: 0.95

## 6.1 PEMILIHAN PARAMETER DENGAN PREPROCESSING

Sekarang katakanlah kita ingin menemukan parameter yang lebih baik untuk SVC menggunakan GridSearchCV, seperti yang dibahas dalam Bab 5. Bagaimana kita harus melakukan ini? Pendekatan naif mungkin terlihat seperti ini:

**In[3]:**

```
from sklearn.model_selection import GridSearchCV
# for illustration purposes only, don't use this code!
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
grid.fit(X_train_scaled, y_train)
print("Best cross-validation accuracy: {:.2f}".format(grid.best_score_))
print("Best set score: {:.2f}".format(grid.score(X_test_scaled, y_test)))
print("Best parameters: ", grid.best_params_)
```

**Out[3]:**

```
Best cross-validation accuracy: 0.98
Best set score: 0.97
Best parameters: {'gamma': 1, 'C': 1}
```

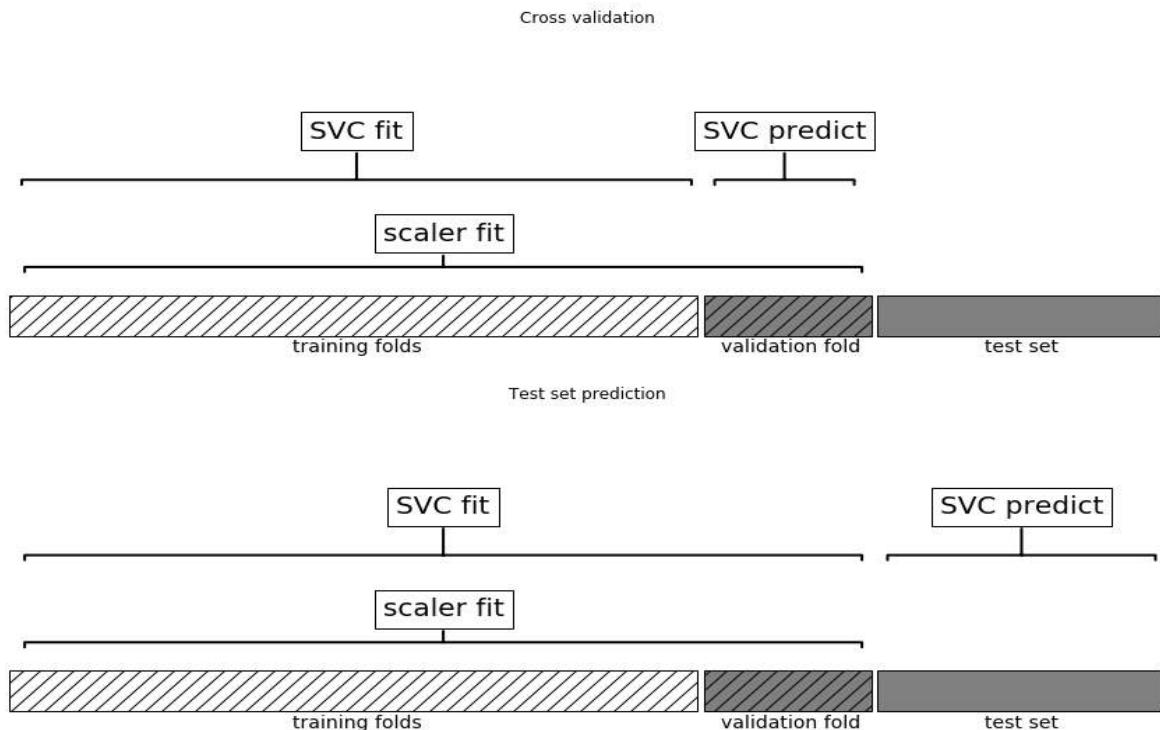
Di sini, kami menjalankan pencarian grid atas parameter SVC menggunakan data yang diskalakan. Namun, ada tangkapan halus dalam apa yang baru saja kita lakukan. Saat menskalakan data, kami menggunakan semua data dalam set pelatihan untuk mengetahui cara melatihnya. Kami kemudian menggunakan data pelatihan yang diskalakan untuk menjalankan pencarian grid kami menggunakan validasi silang.

Untuk setiap pemisahan dalam validasi silang, beberapa bagian dari set pelatihan asli akan dinyatakan sebagai bagian pelatihan dari pemisahan, dan beberapa bagian pengujian dari pemisahan tersebut. Bagian uji digunakan untuk mengukur data baru yang akan terlihat seperti model yang dilatih pada bagian pelatihan. Namun, kami telah menggunakan informasi yang terkandung dalam bagian uji pemisahan, saat menskalakan data. Ingat bahwa bagian pengujian di setiap pemisahan dalam validasi silang adalah bagian dari set pelatihan, dan kami menggunakan informasi dari seluruh set pelatihan untuk menemukan penskalaan data yang tepat.

Ini pada dasarnya berbeda dari tampilan data baru pada model. Jika kita mengamati data baru (misalnya, dalam bentuk set pengujian kita), data ini tidak akan digunakan untuk menskalakan data pelatihan, dan mungkin memiliki minimum dan maksimum yang berbeda dari data pelatihan. Contoh berikut (Gambar 6.1) menunjukkan bagaimana pemrosesan data selama validasi silang dan evaluasi akhir berbeda:

**In[4]:**

```
mglearn.plots.plot_improper_processing()
```



**Gambar 6.1** Penggunaan data saat prapemrosesan di luar loop validasi silang

Jadi, pemisahan dalam validasi silang tidak lagi mencerminkan dengan benar bagaimana data baru akan terlihat pada proses pemodelan. Kami telah membocorkan informasi dari bagian data ini ke dalam proses pemodelan kami. Ini akan menyebabkan hasil yang terlalu optimis selama validasi silang, dan mungkin pemilihan parameter suboptimal.

Untuk mengatasi masalah ini, pemisahan dataset selama validasi silang harus dilakukan sebelum melakukan pra-pemrosesan. Setiap proses yang mengekstrak pengetahuan dari kumpulan data hanya boleh diterapkan ke bagian pelatihan kumpulan data, jadi validasi silang apa pun harus menjadi "loop terluar" dalam pemrosesan Anda.

Untuk mencapai ini di scikit-learn dengan fungsi `cross_val_score` dan fungsi `GridSearchCV`, kita dapat menggunakan kelas Pipeline. Kelas Pipeline adalah kelas yang memungkinkan "menempelkan" beberapa langkah pemrosesan menjadi satu estimator scikit-learn. Kelas Pipeline itu sendiri memiliki metode `fit`, `predict`, dan `skor` dan berperilaku seperti model lainnya di scikit-learn. Kasus penggunaan paling umum dari kelas Pipeline adalah dalam merangkai langkah-langkah prapemrosesan (seperti penskalaan data) bersama dengan model yang diawasi seperti pengklasifikasi.

## 6.2 MEMBANGUN PIPELINE

Mari kita lihat bagaimana kita dapat menggunakan kelas Pipeline untuk mengekspresikan alur kerja untuk melatih SVM setelah menskalakan data dengan `MinMaxScaler` (untuk saat ini tanpa pencarian grid). Pertama, kita bangun objek pipeline dengan memberikan daftar langkah-langkahnya. Setiap langkah adalah tuple yang berisi nama (string apa pun yang Anda pilih) dan turunan dari estimator:

**In[5]:**

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])
```

Di sini, kami membuat dua langkah: yang pertama, disebut "scaler", adalah turunan dari MinMaxScaler, dan yang kedua, disebut "svm", adalah turunan dari SVC. Sekarang, kita dapat menyesuaikan alurnya, seperti estimator scikit-learn lainnya:

**In[6]:**

```
pipe.fit(X_train, y_train)
```

Di sini, pipe.fit pertama memanggil fit pada langkah pertama (scaler), kemudian mentransformasikan data pelatihan menggunakan scaler, dan akhirnya menyesuaikan SVM dengan data yang diskalakan. Untuk mengevaluasi data pengujian, kami cukup memanggil pipe.score:

**In[7]:**

```
print("Test score: {:.2f}".format(pipe.score(X_test, y_test)))
```

**Out[7]:**

```
Test score: 0.95
```

Memanggil metode skor pada pipeline pertama-tama mengubah data pengujian menggunakan scaler, lalu memanggil metode skor pada SVM menggunakan data pengujian yang diskalakan. Seperti yang Anda lihat, hasilnya identik dengan yang kita dapatkan dari kode di awal bab, saat melakukan transformasi dengan tangan. Dengan menggunakan pipeline, kami mengurangi kode yang diperlukan untuk proses "prapemrosesan + klasifikasi". Namun, manfaat utama menggunakan pipeline adalah sekarang kita dapat menggunakan estimator tunggal ini di cross\_val\_score atau GridSearchCV.

### 6.3 MENGGUNAKAN PIPELINE DALAM PENCARIAN GRID

Menggunakan pipeline dalam pencarian grid bekerja dengan cara yang sama seperti menggunakan estimator lainnya. Kami mendefinisikan grid parameter untuk mencari, dan membangun GridSearchCV dari pipa dan grid parameter. Saat menentukan grid parameter, ada sedikit perubahan. Kita perlu menentukan untuk setiap parameter langkah mana dari pipa itu. Kedua parameter yang ingin kita sesuaikan, C dan gamma, adalah parameter SVC, langkah kedua. Kami memberi langkah ini nama "svm". Sintaks untuk mendefinisikan grid parameter untuk pipeline adalah untuk menentukan nama langkah untuk setiap parameter, diikuti oleh (garis bawah ganda), diikuti dengan nama parameter. Untuk mencari parameter C dari SVC karena itu kita harus menggunakan "svm\_C" sebagai kunci dalam kamus parameter grid, dan juga untuk gamma:

**In[8]:**

```
param_grid = {'svm__C': [0.001, 0.01, 0.1, 1, 10, 100],
              'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

Dengan grid parameter ini kita bisa menggunakan GridSearch CV seperti biasa:

**In[9]:**

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation accuracy: {:.2f}".format(grid.best_score_))
print("Test set score: {:.2f}".format(grid.score(X_test, y_test)))
print("Best parameters: {}".format(grid.best_params_))
```

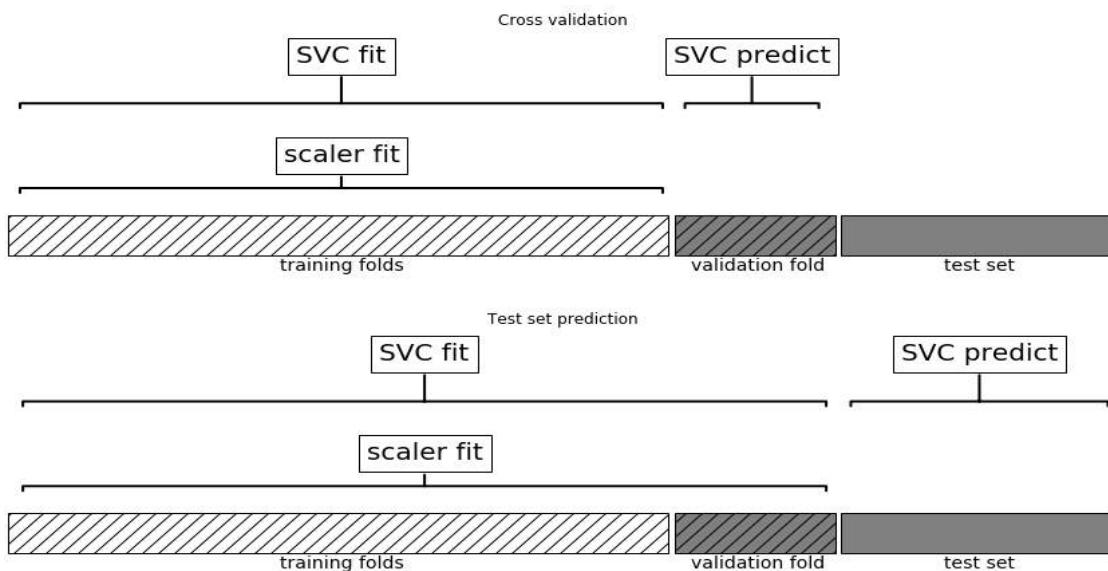
**Out[9]:**

```
Best cross-validation accuracy: 0.98
Test set score: 0.97
Best parameters: {'svm_C': 1, 'svm_gamma': 1}
```

Berbeda dengan pencarian grid yang kami lakukan sebelumnya, sekarang untuk setiap pemisahan dalam validasi silang, MinMaxScaler diperbaiki hanya dengan pemisahan pelatihan dan tidak ada informasi yang bocor dari pemisahan pengujian ke dalam pencarian parameter. Bandingkan ini (Gambar 6.2) dengan Gambar 6.1 sebelumnya dalam bab ini:

**In[10]:**

```
mglearn.plots.plot_proper_processing()
```



**Gambar 6.2** Penggunaan data saat prapemrosesan di dalam loop validasi silang dengan pipeline

Dampak dari kebocoran informasi dalam validasi silang bervariasi tergantung pada sifat dari langkah pra-pemrosesan. Memperkirakan skala data menggunakan lipatan uji biasanya tidak memiliki dampak yang buruk, sementara menggunakan lipatan uji dalam ekstraksi fitur dan pemilihan fitur dapat menyebabkan perbedaan hasil yang substansial.

#### Ilustrasi Kebocoran Informasi

Contoh yang bagus dari kebocoran informasi dalam validasi silang diberikan dalam buku Hastie, Tibshirani, dan Friedman *The Elements of Statistical Learning*, dan kami mereproduksi versi yang diadaptasi di sini. Mari kita pertimbangkan tugas regresi sintetik

dengan 100 sampel dan 1.000 fitur yang diambil sampelnya secara independen dari distribusi Gaussian. Kami juga mengambil sampel respons dari distribusi Gaussian:

**In[11]:**

```
rnd = np.random.RandomState(seed=0)
X = rnd.normal(size=(100, 10000))
y = rnd.normal(size=(100,))
```

Mengingat cara kami membuat dataset, tidak ada hubungan antara data, X, dan target, y (mereka independen), jadi tidak mungkin untuk mempelajari apa pun dari dataset ini. Sekarang kita akan melakukan hal berikut. Pertama, pilih yang paling informatif dari 10 fitur menggunakan pemilihan fitur SelectPercentile, dan kemudian kami mengevaluasi regressor Ridge menggunakan validasi silang:

**In[12]:**

```
from sklearn.feature_selection import SelectPercentile, f_regression

select = SelectPercentile(score_func=f_regression, percentile=5).fit(X, y)
X_selected = select.transform(X)
print("X_selected.shape: {}".format(X_selected.shape))
```

**Out[12]:**

```
X_selected.shape: (100, 500)
```

**In[13]:**

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
print("Cross-validation accuracy (cv only on ridge): {:.2f}".format(
    np.mean(cross_val_score(Ridge(), X_selected, y, cv=5))))
```

**Out[13]:**

```
Cross-validation accuracy (cv only on ridge): 0.91
```

Rata-rata  $R^2$  yang dihitung dengan validasi silang adalah 0,91, menunjukkan model yang sangat baik. Ini jelas tidak benar, karena data kami sepenuhnya acak. Apa yang terjadi di sini adalah bahwa pemilihan fitur kami memilih beberapa fitur di antara 10.000 fitur acak yang (secara kebetulan) berkorelasi sangat baik dengan target. Karena kami cocok dengan pemilihan fitur di luar validasi silang, itu bisa menemukan fitur yang berkorelasi baik pada pelatihan dan lipatan tes. Informasi yang kami bocorkan dari lipatan uji sangat informatif, yang mengarah ke hasil yang sangat tidak realistik. Mari kita bandingkan ini dengan validasi silang yang tepat menggunakan pipeline:

**In[14]:**

```
pipe = Pipeline([('select', SelectPercentile(score_func=f_regression,
                                              percentile=5)),
                 ('ridge', Ridge())])
print("Cross-validation accuracy (pipeline): {:.2f}".format(
    np.mean(cross_val_score(pipe, X, y, cv=5))))
```

**Out[14]:**

```
Cross-validation accuracy (pipeline): -0.25
```

Kali ini, kami mendapatkan skor R<sup>2</sup> negatif, menunjukkan model yang sangat buruk. Menggunakan jalur pipa, pemilihan fitur sekarang berada di dalam loop validasi silang. Ini berarti fitur hanya dapat dipilih menggunakan lipatan pelatihan data, bukan lipatan uji. Seleksi fitur menemukan fitur yang berkorelasi dengan target pada set pelatihan, tetapi karena data sepenuhnya acak, fitur ini tidak berkorelasi dengan target pada set pengujian. Dalam contoh ini, memperbaiki masalah kebocoran data dalam pemilihan fitur membuat perbedaan antara menyimpulkan bahwa model bekerja dengan sangat baik dan menyimpulkan bahwa model tidak berfungsi sama sekali.

#### 6.4 ANTARMUKA PIPELINE

Kelas Pipeline tidak terbatas pada pra-pemrosesan dan klasifikasi, tetapi sebenarnya dapat menggabungkan sejumlah estimator bersama-sama. Misalnya, Anda dapat membangun saluran yang berisi ekstraksi fitur, pemilihan fitur, penskalaan, dan klasifikasi, dengan total empat langkah. Demikian pula, langkah terakhir dapat berupa regresi atau pengelompokan alih-alih klasifikasi.

Satu-satunya persyaratan untuk estimator dalam pipa adalah bahwa semua kecuali langkah terakhir harus memiliki metode transformasi, sehingga mereka dapat menghasilkan representasi baru dari data yang dapat digunakan pada langkah berikutnya. Secara internal, selama panggilan ke Pipeline.fit, panggilan pipa cocok dan kemudian mengubah pada setiap langkah secara bergantian,<sup>2</sup> dengan input yang diberikan oleh output dari metode transformasi dari langkah sebelumnya. Untuk langkah terakhir dalam pipa, hanya fit disebut.

Menyikat beberapa detail yang lebih halus, ini diimplementasikan sebagai berikut. Ingat bahwa pipe.line.steps adalah daftar tupel, jadi pipeline.steps[0][1] adalah estimator pertama, pipe.line.steps[1][1] adalah estimator kedua, dan seterusnya:

**In[15]:**

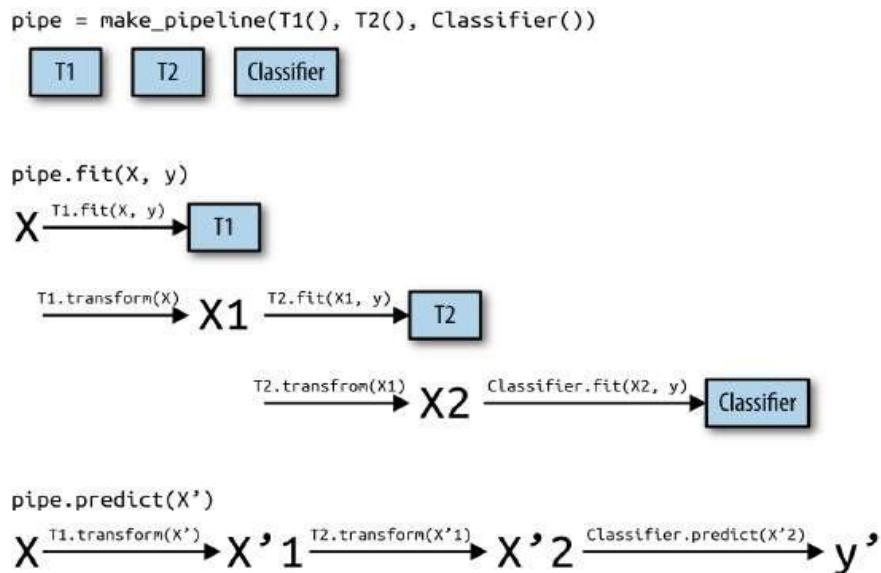
```
def fit(self, X, y):
    X_transformed = X
    for name, estimator in self.steps[:-1]:
        # iterate over all but the final step
        # fit and transform the data
        X_transformed = estimator.fit_transform(X_transformed, y)
    # fit the last step
    self.steps[-1][1].fit(X_transformed, y)
    return self
```

Saat memprediksi menggunakan Pipeline, kami juga mengubah data menggunakan semua kecuali langkah terakhir, lalu memanggil prediksi pada langkah terakhir:

In[16]:

```
def predict(self, X):
    X_transformed = X
    for step in self.steps[:-1]:
        # iterate over all but the final step
        # transform the data
        X_transformed = step[1].transform(X_transformed)
    # fit the last step
    return self.steps[-1][1].predict(X_transformed)
```

Proses tersebut diilustrasikan pada Gambar 6.3 untuk dua transformator, T1 dan T2, dan sebuah pengklasifikasi (disebut Pengklasifikasi).



Gambar 6.3 Ikhtisar pelatihan pipa dan proses prediksi

Pipa sebenarnya bahkan lebih umum dari ini. Tidak ada persyaratan untuk langkah terakhir dalam pipeline untuk memiliki fungsi prediksi, dan kita dapat membuat pipeline yang hanya berisi, misalnya, scaler dan PCA. Kemudian, karena langkah terakhir (PCA) memiliki metode transformasi, kita dapat memanggil transformasi pada pipa untuk mendapatkan output dari PCA.transform diterapkan pada data yang diproses oleh langkah sebelumnya. Langkah terakhir dari pipa hanya diperlukan untuk memiliki metode yang cocok.

## 6.5 PEMBUATAN PIPELINE

Membuat pipeline menggunakan sintaks yang dijelaskan sebelumnya terkadang agak rumit, dan kita sering kali tidak memerlukan nama yang ditentukan pengguna untuk setiap langkah. Ada fungsi kemudahan, `make_pipeline`, yang akan membuat pipeline untuk kita dan secara otomatis memberi nama setiap langkah berdasarkan kelasnya. Sintaks untuk `make_pipeline` adalah sebagai berikut:

**In[17]:**

```
from sklearn.pipeline import make_pipeline
# standard syntax
pipe_long = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC(C=100))])
# abbreviated syntax
pipe_short = make_pipeline(MinMaxScaler(), SVC(C=100))
```

Objek pipa pipe\_long dan pipe\_short melakukan hal yang persis sama, tetapi pipe\_short memiliki langkah-langkah yang diberi nama secara otomatis. Kita bisa melihat nama-nama langkah dengan melihat atribut langkah:

**In[18]:**

```
print("Pipeline steps:\n{}".format(pipe_short.steps))
```

**Out[18]:**

```
Pipeline steps:
[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))),
 ('svc', SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape=None, degree=3, gamma='auto',
             kernel='rbf', max_iter=-1, probability=False,
             random_state=None, shrinking=True, tol=0.001,
             verbose=False))]
```

Langkah-langkahnya diberi nama minmaxscaler dan svc. Secara umum, nama langkah hanyalah versi huruf kecil dari nama kelas. Jika beberapa langkah memiliki kelas yang sama, angka ditambahkan:

**In[19]:**

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

pipe = make_pipeline(StandardScaler(), PCA(n_components=2), StandardScaler())
print("Pipeline steps:\n{}".format(pipe.steps))
```

**Out[19]:**

```
Pipeline steps:
[('standardscaler-1', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('pca', PCA(copy=True, iterated_power=4, n_components=2, random_state=None,
             svd_solver='auto', tol=0.0, whiten=False)),
 ('standardscaler-2', StandardScaler(copy=True, with_mean=True, with_std=True))]
```

Seperti yang Anda lihat, langkah StandardScaler pertama diberi nama standardscaler-1 dan kedua standardscaler-2. Namun, dalam pengaturan seperti itu mungkin lebih baik menggunakan konstruksi Pipeline dengan nama eksplisit, untuk memberikan lebih banyak nama semantik untuk setiap langkah.

## 6.6 MENGAKSES ATRIBUT PIPELINE

Seringkali Anda ingin memeriksa atribut dari salah satu langkah pipeline—misalnya, koefisien model linier atau komponen yang diekstraksi oleh PCA. Cara termudah untuk mengakses langkah-langkah dalam pipa adalah melalui atribut named\_steps, yang merupakan kamus dari nama langkah ke penaksir:

**In[20]:**

```
# fit the pipeline defined before to the cancer dataset
pipe.fit(cancer.data)
# extract the first two principal components from the "pca" step
components = pipe.named_steps["pca"].components_
print("components.shape: {}".format(components.shape))
```

**Out[20]:**

```
components.shape: (2, 30)
```

## 6.7 MENGAKSES ATRIBUT DALAM PIPA YANG DICARI GRID

Seperti yang telah kita bahas sebelumnya dalam bab ini, salah satu alasan utama menggunakan pipeline adalah untuk melakukan pencarian grid. Tugas umum adalah mengakses beberapa langkah pipeline di dalam pencarian grid. Mari kita cari pengklasifikasi LogisticRegression pada kumpulan data kanker, menggunakan Pipeline dan StandardScaler untuk menskalakan data sebelum meneruskannya ke pengklasifikasi cRegression Logisti. Pertama kita membuat pipeline menggunakan fungsi make\_pipeline:

**In[21]:**

```
from sklearn.linear_model import LogisticRegression

pipe = make_pipeline(StandardScaler(), LogisticRegression())
```

Selanjutnya, kita membuat grid parameter. Seperti yang dijelaskan di Bab 2, parameter regularisasi yang akan disetel untuk LogisticRegression adalah parameter C. Kami menggunakan grid logaritmik untuk parameter ini, mencari antara 0,01 dan 100. Karena kami menggunakan fungsi make\_pipeline, nama langkah LogisticRegression dalam pipeline adalah nama kelas dengan huruf kecil, regresi logistik. Untuk menyetel parameter C, oleh karena itu kita harus menentukan grid parameter untuk regresi logistik C:

**In[22]:**

```
param_grid = {'logisticregression_C': [0.01, 0.1, 1, 10, 100]}
```

Seperti biasa, kami membagi dataset kanker menjadi set pelatihan dan pengujian, dan menyesuaikan pencarian grid:

**In[23]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=4)
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
```

Jadi bagaimana kita mengakses koefisien model LogisticRegression terbaik yang ditemukan oleh GridSearchCV? Dari Bab 5 kita tahu bahwa model terbaik yang ditemukan oleh GridSearchCV, dilatih pada semua data pelatihan, disimpan di grid.best\_estimator\_:

In[24]:

```
print("Best estimator:\n{}".format(grid.best_estimator_))
```

Out[24]:

```
Best estimator:
Pipeline(steps=[('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('logisticregression', LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False))])
```

Best\_estimator\_ ini dalam kasus kami adalah saluran dengan dua langkah, penskala standar dan regresi logistik. Untuk mengakses langkah logisticregression, kita dapat menggunakan atribut named\_steps dari pipeline, seperti yang dijelaskan sebelumnya:

In[25]:

```
print("Logistic regression step:\n{}".format(
    grid.best_estimator_.named_steps["logisticregression"]))
```

Out[25]:

```
Logistic regression step:
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Sekarang setelah kita memiliki instance LogisticRegression yang terlatih, kita dapat mengakses koefisien (bobot) yang terkait dengan setiap fitur input:

In[26]:

```
print("Logistic regression coefficients:\n{}".format(
    grid.best_estimator_.named_steps["logisticregression"].coef_))
```

Out[26]:

```
Logistic regression coefficients:
[[ -0.389 -0.375 -0.376 -0.396 -0.115  0.017 -0.355 -0.39 -0.058  0.209
   -0.495 -0.004 -0.371 -0.383 -0.045  0.198  0.004 -0.049  0.21   0.224
   -0.547 -0.525 -0.499 -0.515 -0.393 -0.123 -0.388 -0.417 -0.325 -0.139]]
```

Ini mungkin ekspresi yang agak panjang, tetapi sering kali berguna dalam memahami model Anda.

## 6.8 LANGKAH PEMROSESAN DALAM ALUR KERJA MACHINE LEARNING

Dengan menggunakan pipeline, kami dapat merangkum semua langkah pemrosesan dalam alur kerja machine learning kami dalam satu estimator scikit-learn. Manfaat lain dari melakukan ini adalah kita sekarang dapat menyesuaikan parameter prapemrosesan menggunakan hasil tugas yang diawasi seperti regresi atau klasifikasi. Dalam bab sebelumnya, kami menggunakan fitur polinomial pada dataset boston sebelum menerapkan ridge

regressor. Mari kita modelkan itu menggunakan pipa sebagai gantinya. Pipeline berisi tiga langkah—menskalakan data, menghitung fitur polinomial, dan regresi ridge:

**In[27]:**

```
from sklearn.datasets import load_boston
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target,
                                                    random_state=0)

from sklearn.preprocessing import PolynomialFeatures
pipe = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(),
    Ridge())
```

Bagaimana kita tahu derajat polinomial mana yang harus dipilih, atau apakah akan memilih polinomial atau interaksi sama sekali? Idealnya kita ingin memilih parameter derajat berdasarkan hasil klasifikasi. Menggunakan pipa kami, kami dapat mencari parameter derajat bersama-sama dengan parameter alpha Ridge. Untuk melakukan ini, kami mendefinisikan param\_grid yang berisi keduanya, yang diawali dengan tepat dengan nama langkah:

**In[28]:**

```
param_grid = {'polynomialfeatures_degree': [1, 2, 3],
              'ridge_alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
```

Sekarang kita dapat menjalankan pencarian grid kita lagi:

**In[29]:**

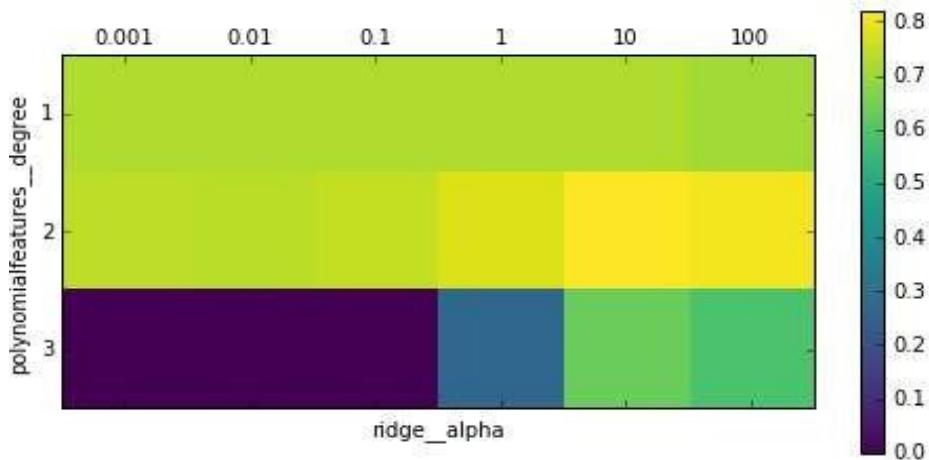
```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1)
grid.fit(X_train, y_train)
```

Kami dapat memvisualisasikan hasil validasi silang menggunakan peta panas (Gambar 6.4), seperti yang kami lakukan di Bab 5:

**In[30]:**

```
plt.matshow(grid.cv_results_['mean_test_score'].reshape(3, -1),
            vmin=0, cmap="viridis")
plt.xlabel("ridge_alpha")
plt.ylabel("polynomialfeatures_degree")
plt.xticks(range(len(param_grid['ridge_alpha'])), param_grid['ridge_alpha'])
plt.yticks(range(len(param_grid['polynomialfeatures_degree'])),
           param_grid['polynomialfeatures_degree'])

plt.colorbar()
```



**Gambar 6.4** Peta panas skor validasi silang rata-rata sebagai fungsi derajat fitur polinomial dan parameter alfa Ridge

Melihat hasil yang dihasilkan oleh validasi silang, kita dapat melihat bahwa menggunakan polinomial derajat dua membantu, tetapi polinomial derajat tiga jauh lebih buruk daripada derajat satu atau dua. Ini tercermin dalam parameter terbaik yang ditemukan:

**In[31]:**

```
print("Best parameters: {}".format(grid.best_params_))
```

**Out[31]:**

```
Best parameters: {'polynomialfeatures_degree': 2, 'ridge_alpha': 10}
```

Yang mengarah ke skor berikut:

**In[32]:**

```
print("Test-set score: {:.2f}".format(grid.score(X_test, y_test)))
```

**Out[32]:**

```
Test-set score: 0.77
```

Mari kita jalankan pencarian grid tanpa fitur polinomial untuk perbandingan:

**In[33]:**

```
param_grid = {'ridge_alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
pipe = make_pipeline(StandardScaler(), Ridge())
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
print("Score without poly features: {:.2f}".format(grid.score(X_test, y_test)))
```

**Out[33]:**

```
Score without poly features: 0.63
```

Seperti yang kita harapkan melihat hasil pencarian grid divisualisasikan pada Gambar 6.4, tidak menggunakan fitur polinomial mengarah ke hasil yang jelas lebih buruk.

Mencari parameter pra-pemrosesan bersama dengan parameter model adalah strategi yang sangat kuat. Namun, perlu diingat bahwa GridSearchCV mencoba semua kemungkinan kombinasi dari parameter yang ditentukan. Oleh karena itu, menambahkan lebih banyak parameter ke grid Anda secara eksponensial meningkatkan jumlah model yang perlu dibangun.

## 6.9 PENCARIAN KOTAK MODEL MANA YANG DIGUNAKAN

Anda bahkan dapat melangkah lebih jauh dalam menggabungkan GridSearchCV dan Pipeline: Anda juga dapat menelusuri langkah-langkah aktual yang dilakukan di dalam pipeline (misalnya apakah akan menggunakan StandardScaler atau MinMaxScaler). Ini mengarah ke ruang pencarian yang lebih besar dan harus dipertimbangkan dengan hati-hati. Mencoba semua solusi yang mungkin biasanya bukan strategi pembelajaran mesin yang layak. Namun, berikut adalah contoh membandingkan RandomForest Classifier dan SVC pada dataset iris. Kami tahu bahwa SVC mungkin memerlukan data untuk diskalakan, jadi kami juga mencari apakah akan menggunakan StandardScaler atau tanpa pra-pemrosesan. Untuk RandomForestClassifier, kita tahu bahwa tidak diperlukan preprocessing. Kita mulai dengan mendefinisikan pipa. Di sini, kami secara eksplisit menyebutkan langkah-langkahnya. Kami ingin dua langkah, satu untuk preprocessing dan kemudian classifier. Kami dapat membuat instance ini menggunakan SVC dan StandardScaler:

In[34]:

```
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVC())])
```

Sekarang kita dapat menentukan parameter\_grid untuk mencari. Kami ingin pengklasifikasi menjadi RandomForestClassifier atau SVC. Karena mereka memiliki parameter yang berbeda untuk disetel, dan memerlukan prapemrosesan yang berbeda, kita dapat menggunakan daftar kisi-kisi pencarian yang telah kita diskusikan di “Mencari ruang yang bukan kisi-kisi” pada halaman 271. Untuk menetapkan penduga ke suatu langkah, kita gunakan nama langkah sebagai nama parameter. Saat kita ingin melewati satu langkah dalam pipeline (misalnya, karena kita tidak memerlukan pra-pemrosesan untuk RandomForest), kita dapat menyetel langkah itu ke None:

In[35]:

```
from sklearn.ensemble import RandomForestClassifier

param_grid = [
    {'classifier': [SVC()], 'preprocessing': [StandardScaler(), None],
     'classifier_gamma': [0.001, 0.01, 0.1, 1, 10, 100],
     'classifier_C': [0.001, 0.01, 0.1, 1, 10, 100]}, 
    {'classifier': [RandomForestClassifier(n_estimators=100)],
     'preprocessing': [None], 'classifier_max_features': [1, 2, 3]}]
```

Sekarang kita dapat membuat instance dan menjalankan pencarian grid seperti biasa, di sini di dataset kanker:

**In[36]:**

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best params:\n{}\n".format(grid.best_params_))
print("Best cross-validation score: {:.2f}\n".format(grid.best_score_))
print("Test-set score: {:.2f}\n".format(grid.score(X_test, y_test)))
```

**Out[36]:**

```
Best params:
{'classifier':
 SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
      max_iter=-1, probability=False, random_state=None, shrinking=True,
      tol=0.001, verbose=False),
 'preprocessing':
 StandardScaler(copy=True, with_mean=True, with_std=True),
 'classifier__C': 10, 'classifier__gamma': 0.01}

Best cross-validation score: 0.99
Test-set score: 0.98
```

Hasil dari pencarian grid adalah SVC dengan preprocessing StandardScaler, C=10, dan gamma=0.01 memberikan hasil terbaik.

## 6.10 RINGKASAN DAN PANDANGAN

Dalam bab ini, kami memperkenalkan kelas Pipeline, alat serba guna untuk menyatukan beberapa langkah pemrosesan dalam alur kerja machine learning. Aplikasi pembelajaran mesin di dunia nyata jarang melibatkan penggunaan model yang terisolasi, dan sebaliknya merupakan urutan langkah pemrosesan. Menggunakan pipeline memungkinkan kita untuk merangkum beberapa langkah ke dalam satu objek Python yang mematuhi antarmuka scikit-learn fit, predict, dan transform yang sudah dikenal. Khususnya ketika melakukan evaluasi model menggunakan validasi silang dan pemilihan parameter menggunakan pencarian grid, menggunakan kelas Pipeline untuk menangkap semua langkah pemrosesan sangat penting untuk evaluasi yang tepat.

Kelas Pipeline juga memungkinkan penulisan kode yang lebih ringkas, dan mengurangi kemungkinan kesalahan yang dapat terjadi saat membangun rantai pemrosesan tanpa kelas pipeline (seperti lupa menerapkan semua transformator pada set pengujian, atau tidak menerapkannya dalam urutan yang benar). Memilih kombinasi yang tepat dari ekstraksi fitur, preprocessing, dan model adalah suatu seni, dan sering kali membutuhkan beberapa trial and error.

Namun, dengan menggunakan jalur pipa, “mencoba” banyak langkah pemrosesan yang berbeda ini cukup sederhana. Saat berekspeten, berhati-hatilah untuk tidak terlalu memperumit proses Anda, dan pastikan untuk mengevaluasi apakah setiap komponen yang Anda sertakan dalam model Anda diperlukan.

Dengan bab ini, kami telah menyelesaikan survei alat dan algoritma tujuan umum yang disediakan oleh scikit-learn. Anda sekarang memiliki semua keterampilan yang diperlukan dan mengetahui mekanisme yang diperlukan untuk menerapkan pembelajaran mesin dalam

praktik. Dalam bab berikutnya, kita akan menyelam lebih dalam ke satu jenis data tertentu yang umum terlihat dalam praktik, dan yang memerlukan keahlian khusus untuk menangani dengan benar: data teks.

## BAB 7

### BEKERJA DENGAN DATA TEKS

Dalam Bab 7, kita berbicara tentang dua jenis fitur yang dapat mewakili properti data: fitur berkelanjutan yang menggambarkan kuantitas, dan fitur kategoris yang merupakan item dari daftar tetap. Ada jenis fitur ketiga yang dapat ditemukan di banyak aplikasi, yaitu teks. Misalnya, jika kita ingin mengklasifikasikan sebuah pesan email sebagai email yang sah atau spam, isi email tersebut tentunya akan berisi informasi penting untuk tugas klasifikasi ini. Atau mungkin kita ingin belajar tentang pendapat seorang politikus tentang topik keimigrasian. Di sini, pidato atau tweet individu tersebut dapat memberikan informasi yang berguna. Dalam layanan pelanggan, kita sering ingin mengetahui apakah sebuah pesan adalah keluhan atau pertanyaan. Kami dapat menggunakan baris subjek dan konten pesan untuk secara otomatis menentukan maksud pelanggan, yang memungkinkan kami untuk mengirim pesan ke departemen yang sesuai, atau bahkan mengirim balasan yang sepenuhnya otomatis.

Data teks biasanya direpresentasikan sebagai string, terdiri dari karakter. Dalam salah satu contoh yang baru saja diberikan, panjang data teks akan bervariasi. Fitur ini jelas sangat berbeda dari fitur numerik yang telah kita bahas sejauh ini, dan kita perlu memproses data sebelum dapat menerapkan algoritme pembelajaran mesin kita ke dalamnya.

#### **7.1 JENIS DATA DIREPRESENTASIKAN SEBAGAI STRING**

Sebelum kita menyelami langkah-langkah pemrosesan yang mewakili data teks untuk pembelajaran mesin, kami ingin membahas secara singkat berbagai jenis data teks yang mungkin Anda temui. Teks biasanya hanya berupa string dalam kumpulan data Anda, tetapi tidak semua fitur string harus diperlakukan sebagai teks. Sebuah fitur string terkadang dapat mewakili variabel kategoris, seperti yang telah kita bahas di Bab 5. Tidak ada cara untuk mengetahui bagaimana memperlakukan fitur string sebelum melihat data.

Ada empat jenis data string yang mungkin Anda lihat:

- Kategori data
- String gratis yang dapat dipetakan secara semantik ke kategori
- Data string terstruktur
- Data teks

Data kategoris adalah data yang berasal dari daftar tetap. Katakanlah Anda mengumpulkan data melalui survei di mana Anda menanyakan warna favorit orang-orang, dengan menu tarik-turun yang memungkinkan mereka memilih dari "merah", "hijau", "biru", "kuning", "hitam", "putih," "ungu", dan "merah muda". Ini akan menghasilkan kumpulan data dengan tepat delapan kemungkinan nilai yang berbeda, yang dengan jelas mengkodekan variabel kategoris. Anda dapat memeriksa apakah ini kasus data Anda dengan mengamatinya (jika Anda melihat sangat banyak string yang berbeda, kecil kemungkinannya bahwa ini adalah variabel kategoris) dan mengonfirmasinya dengan menghitung nilai unik pada kumpulan data, dan mungkin histogram tentang seberapa sering masing-masing muncul. Anda juga mungkin ingin memeriksa apakah setiap variabel benar-benar sesuai dengan kategori yang masuk akal

untuk aplikasi Anda. Mungkin di tengah-tengah keberadaan survei Anda, seseorang menemukan bahwa "hitam" salah eja sebagai "blak" dan kemudian memperbaiki survei tersebut. Akibatnya, kumpulan data Anda berisi "blak" dan "hitam", yang sesuai dengan makna semantik yang sama dan harus dikonsolidasikan.

Sekarang bayangkan alih-alih menyediakan menu tarik-turun, Anda menyediakan bidang teks bagi pengguna untuk memberikan warna favorit mereka sendiri. Banyak orang mungkin merespons dengan nama warna seperti "hitam" atau "biru". Orang lain mungkin membuat kesalahan ketik, menggunakan ejaan yang berbeda seperti "abu-abu" dan "abu-abu", atau menggunakan nama yang lebih menggugah dan spesifik seperti "biru tengah malam". Anda juga akan memiliki beberapa entri yang sangat aneh. Beberapa contoh bagus datang dari Survei Warna xkcd, di mana orang harus memberi nama warna dan muncul dengan nama seperti "velociraptor cloaka" dan "oranye kantor dokter gigi saya. Saya masih ingat ketomenya perlahan-lahan berhembus ke mulut saya yang menganga," yang sulit dipetakan ke warna secara otomatis (atau sama sekali).

Respon yang dapat Anda peroleh dari bidang teks termasuk dalam kategori kedua dalam daftar, string bebas yang dapat dipetakan secara semantik ke kategori. Mungkin akan lebih baik untuk mengkodekan data ini sebagai variabel kategori, di mana Anda dapat memilih kategori baik dengan menggunakan entri yang paling umum, atau dengan mendefinisikan kategori yang akan menangkap tanggapan dengan cara yang masuk akal untuk aplikasi Anda. Anda kemudian mungkin memiliki beberapa kategori untuk warna standar, mungkin kategori "berwarna-warni" untuk orang-orang yang memberikan jawaban seperti "garis-garis hijau dan merah," dan kategori "lainnya" untuk hal-hal yang tidak dapat dikodekan sebaliknya. Pemrosesan awal string semacam ini dapat membutuhkan banyak upaya manual dan tidak mudah otomatis. Jika Anda berada dalam posisi di mana Anda dapat mempengaruhi pengumpulan data, kami sangat menyarankan untuk menghindari nilai yang dimasukkan secara manual untuk konsep yang lebih baik ditangkap menggunakan variabel kategori.

Seringkali, nilai yang dimasukkan secara manual tidak sesuai dengan kategori tetap, tetapi masih memiliki beberapa struktur dasar, seperti alamat, nama tempat atau orang, tanggal, nomor telepon, atau pengenal lainnya. Jenis string ini seringkali sangat sulit untuk diuraikan, dan perlakunya sangat bergantung pada konteks dan domain. Perlakuan sistematis atas kasus-kasus ini berada di luar cakupan buku ini.

Kategori terakhir dari data string adalah data teks bentuk bebas yang terdiri dari frase atau kalimat. Contohnya termasuk tweet, log obrolan, dan ulasan hotel, serta kumpulan karya Shakespeare, konten Wikipedia, atau kumpulan 50.000 eBook Project Gutenberg. Semua koleksi ini sebagian besar berisi informasi sebagai kalimat yang terdiri dari kata-kata.<sup>1</sup> Demi kesederhanaan, mari kita asumsikan semua dokumen kita dalam satu bahasa, Inggris.<sup>2</sup> Dalam konteks analisis teks, kumpulan data sering disebut corpus, dan masing-masing titik data, direpresentasikan sebagai teks tunggal, disebut dokumen. Istilah-istilah ini berasal dari komunitas pencarian informasi (IR) dan pemrosesan bahasa alami (NLP), yang keduanya sebagian besar berurusan dengan data teks.

## 7.2 CONTOH APLIKASI: ANALISIS REVIEW FILM

Sebagai contoh berjalan dalam bab ini, kami akan menggunakan kumpulan data ulasan film dari situs web IMDb (Database Film Internet) yang dikumpulkan oleh peneliti Stanford Andrew Maas.<sup>3</sup> Kumpulan data ini berisi teks ulasan, bersama dengan label yang menunjukkan apakah ulasan itu "positif" atau "negatif". Situs web IMDb sendiri berisi peringkat dari 1 hingga 10. Untuk menyederhanakan pemodelan, anotasi ini diringkas sebagai kumpulan data klasifikasi dua kelas di mana ulasan dengan skor 6 atau lebih tinggi diberi label sebagai positif, dan sisanya sebagai negatif. Kami akan membiarkan pertanyaan apakah ini representasi data yang baik, dan cukup gunakan data yang disediakan oleh Andrew Maas.

Setelah membongkar data, kumpulan data disediakan sebagai file teks dalam dua folder terpisah, satu untuk data pelatihan dan satu untuk data pengujian. Masing-masing pada gilirannya memiliki dua subfolder, satu disebut pos dan satu lagi disebut neg:

**In[2]:**

```
!tree -L 2 data/aclImdb
```

**Out[2]:**

```
data/aclImdb
└── test
    ├── neg
    └── pos
└── train
    ├── neg
    └── pos
```

```
6 directories, 0 files
```

Folder pos berisi semua ulasan positif, masing-masing sebagai file teks terpisah, dan demikian pula untuk folder neg. Ada fungsi pembantu di scikit-learn untuk memuat file yang disimpan dalam struktur folder seperti itu, di mana setiap subfolder sesuai dengan label, yang disebut `load_files`. Kami menerapkan fungsi `load_files` terlebih dahulu ke data pelatihan:

**In[3]:**

```
from sklearn.datasets import load_files

reviews_train = load_files("data/aclImdb/train/")
# load_files returns a bunch, containing training texts and training labels
text_train, y_train = reviews_train.data, reviews_train.target
print("type of text_train: {}".format(type(text_train)))
print("length of text_train: {}".format(len(text_train)))
print("text_train[1]:\n{}".format(text_train[1]))
```

**Out[3]:**

```
type of text_train: <class 'list'>
length of text_train: 25000
text_train[1]:
b'Words can\'t describe how bad this movie is. I can\'t explain it by writing
only. You have too see it for yourself to get at grip of how horrible a movie
really can be. Not that I recommend you to do that. There are so many
clich\xc3\x9as, mistakes (and all other negative things you can imagine) here
that will just make you cry. To start with the technical first, there are a
LOT of mistakes regarding the airplane. I won\'t list them here, but just
mention the coloring of the plane. They didn\'t even manage to show an
airliner in the colors of a fictional airline, but instead used a 747
painted in the original Boeing livery. Very bad. The plot is stupid and has
been done many times before, only much, much better. There are so many
ridiculous moments here that i lost count of it really early. Also, I was on
the bad guys\' side all the time in the movie, because the good guys were so
stupid. "Executive Decision" should without a doubt be you\'re choice over
this one, even the "Turbulence"-movies are better. In fact, every other
movie in the world is better than this one.'
```

Anda dapat melihat bahwa `text_train` adalah daftar dengan panjang 25.000, di mana setiap entri adalah string yang berisi ulasan. Kami mencetak ulasan dengan indeks 1. Anda juga dapat melihat bahwa ulasan berisi beberapa jeda baris HTML (`<br />`). Meskipun ini tidak mungkin berdampak besar pada model pembelajaran mesin kami, lebih baik untuk membersihkan data dan menghapus pemformatan ini sebelum kami melanjutkan:

**In[4]:**

```
text_train = [doc.replace(b"<br />", b" ") for doc in text_train]
```

Jenis entri `text_train` akan bergantung pada versi Python Anda. Dalam Python 3, mereka akan bertipe byte yang mewakili pengkodean biner dari data string. Dalam Python 2, `text_train` berisi string. Kami tidak akan membahas detail jenis string yang berbeda dalam Python di sini, tetapi kami menyarankan Anda membaca dokumentasi Python 2 dan/atau Python 3 mengenai string dan Unicode.

Dataset dikumpulkan sedemikian rupa sehingga kelas positif dan kelas negatif seimbang, sehingga string positif sebanyak negatif:

**In[5]:**

```
print("Samples per class (training): {}".format(np.bincount(y_train)))
```

**Out[5]:**

```
Samples per class (training): [12500 12500]
```

Kami memuat kumpulan data pengujian dengan cara yang sama:

**In[6]:**

```
reviews_test = load_files("data/aclImdb/test/")
text_test, y_test = reviews_test.data, reviews_test.target
print("Number of documents in test data: {}".format(len(text_test)))
print("Samples per class (test): {}".format(np.bincount(y_test)))
text_test = [doc.replace(b"<br />", b" ") for doc in text_test]
```

**Out[6]:**

```
Number of documents in test data: 25000
Samples per class (test): [12500 12500]
```

Tugas yang ingin kami selesaikan adalah sebagai berikut: diberikan ulasan, kami ingin menetapkan label "positif" atau "negatif" berdasarkan konten teks ulasan. Ini adalah tugas klasifikasi biner standar. Namun, data teks tidak dalam format yang dapat ditangani oleh model pembelajaran mesin. Kita perlu mengubah representasi string dari teks menjadi representasi numerik yang dapat kita terapkan pada algoritma pembelajaran mesin kita.

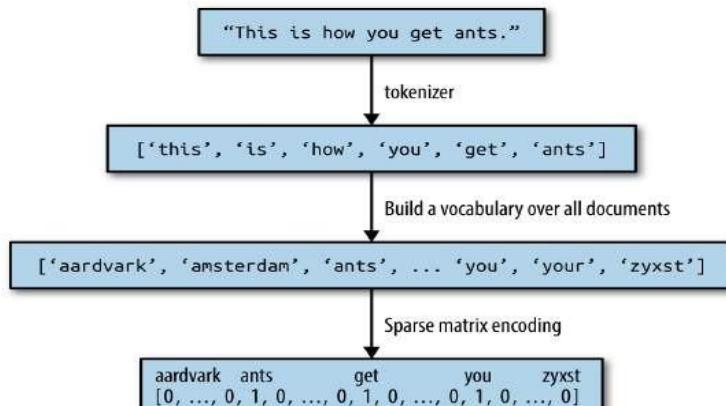
### 7.3 MEWAKILI DATA TEKS SEBAGAI *BAG-OF-WORDS*

Salah satu cara paling sederhana namun efektif dan umum digunakan untuk merepresentasikan teks untuk pembelajaran mesin adalah menggunakan representasi bag-of-words. Saat menggunakan representasi ini, kami membuang sebagian besar struktur teks input, seperti bab, paragraf, kalimat, dan pemformatan, dan hanya menghitung seberapa sering setiap kata muncul di setiap teks dalam korpus. Membuang struktur dan hanya menghitung kemunculan kata mengarah pada citra mental yang merepresentasikan teks sebagai "kantong".

Menghitung representasi bag-of-words untuk korpus dokumen terdiri dari tiga langkah berikut:

1. Tokenisasi. Pisahkan setiap dokumen menjadi kata-kata yang muncul di dalamnya (disebut token), misalnya dengan memisahkannya pada spasi dan tanda baca.
2. Membangun kosakata. Kumpulkan kosakata dari semua kata yang muncul di salah satu dokumen, dan beri nomor (katakanlah, dalam urutan abjad).
3. Pengkodean. Untuk setiap dokumen, hitung seberapa sering setiap kata dalam kosa kata muncul dalam dokumen ini.

Ada beberapa seluk-beluk yang terlibat dalam langkah 1 dan langkah 2, yang akan kita bahas secara lebih rinci nanti dalam bab ini. Untuk saat ini, mari kita lihat bagaimana kita dapat menerapkan pemrosesan bag-of-words menggunakan scikit-learn. Gambar 7.1 mengilustrasikan proses pada string "Beginilah cara Anda mendapatkan semut.". Outputnya adalah satu vektor jumlah kata untuk setiap dokumen. Untuk setiap kata dalam kosakata, kami menghitung seberapa sering kata itu muncul di setiap dokumen. Itu berarti representasi numerik kami memiliki satu fitur untuk setiap kata unik di seluruh kumpulan data. Perhatikan bagaimana urutan kata dalam string asli sama sekali tidak relevan dengan representasi fitur bag-of-words.



**Gambar 7.1** Pemrosesan sekantong kata

## 7.4 MENERAPKAN BAG-OF-WORDS KE DATASET MAINAN

Representasi bag-of-words diimplementasikan di CountVectorizer, yang merupakan transformator. Mari kita terapkan dulu ke kumpulan data mainan, yang terdiri dari dua sampel, untuk melihatnya berfungsi:

**In[7]:**

```
bards_words = ["The fool doth think he is wise,",
               "but the wise man knows himself to be a fool"]
```

Kami mengimpor dan membuat instance CountVectorizer dan menyesuaikannya dengan data mainan kami sebagai berikut:

**In[8]:**

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
vect.fit(bards_words)
```

Pemasangan CountVectorizer terdiri dari tokenisasi data pelatihan dan pembangunan kosakata, yang dapat kita akses sebagai atribut vocabulary\_:

**In[9]:**

```
print("Vocabulary size: {}".format(len(vect.vocabulary_)))
print("Vocabulary content:\n {}".format(vect.vocabulary_))
```

**Out[9]:**

```
Vocabulary size: 13
Vocabulary content:
{'the': 9, 'himself': 5, 'wise': 12, 'he': 4, 'doth': 2, 'to': 11, 'knows': 7,
 'man': 8, 'fool': 3, 'is': 6, 'be': 0, 'think': 10, 'but': 1}
```

Kosakata terdiri dari 13 kata, dari "menjadi" hingga "bijaksana". Untuk membuat representasi bag-of-words untuk data pelatihan, kami menyebutnya transformasi metode:

**In[10]:**

```
bag_of_words = vect.transform(bards_words)
print("bag_of_words: {}".format(repr(bag_of_words)))
```

**Out[10]:**

```
bag_of_words: <2x13 sparse matrix of type '<class 'numpy.int64'>'>
with 16 stored elements in Compressed Sparse Row format>
```

Representasi *bag-of-words* disimpan dalam matriks sparse SciPy yang hanya menyimpan entri yang bukan nol (lihat Bab 1). Matriksnya berbentuk  $2 \times 13$ , dengan satu baris untuk masing-masing dari dua titik data dan satu fitur untuk setiap kata dalam kosakata. Matriks jarang digunakan karena sebagian besar dokumen hanya berisi sebagian kecil kata dalam kosakata, artinya sebagian besar entri dalam larik fitur adalah 0. Pikirkan tentang berapa banyak kata berbeda yang mungkin muncul dalam ulasan film dibandingkan dengan semua kata dalam bahasa Inggris bahasa (yang merupakan model kosa kata). Menyimpan semua angka nol itu akan menjadi penghalang, dan membuang-buang memori. Untuk melihat

isi sebenarnya dari matriks sparse, kita dapat mengubahnya menjadi array NumPy "padat" (yang juga menyimpan semua 0 entri) menggunakan metode toarray:

**In[11]:**

```
print("Dense representation of bag_of_words:\n{}".format(
    bag_of_words.toarray()))
```

**Out[11]:**

```
Dense representation of bag_of_words:
[[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

Kita dapat melihat bahwa jumlah kata untuk setiap kata adalah 0 atau 1; tak satu pun dari dua string di `bag_of_words` berisi kata dua kali. Mari kita lihat cara membaca vektor fitur ini. String pertama ("Orang bodoh menganggap dirinya bijaksana,") dilambangkan sebagai baris pertama, dan berisi kata pertama dalam kosakata, "menjadi", nol kali. Ini juga mengandung kata kedua dalam kosakata, "tetapi", nol kali. Itu berisi kata ketiga, "doth", sekali, dan seterusnya. Melihat kedua baris, kita dapat melihat bahwa kata keempat, "bodoh", kata kesepuluh, "yang", dan kata ketiga belas, "bijaksana", muncul di kedua string.

## 7.5 BAG-OF-WORDS UNTUK ULASAN FILM

Sekarang setelah kita melalui proses bag-of-words secara mendetail, mari kita terapkan pada tugas analisis sentimen untuk ulasan film. Sebelumnya, kami memuat data pelatihan dan pengujian kami dari ulasan IMDb ke dalam daftar string (`text_train` dan `text_test`), yang sekarang akan kami proses:

**In[12]:**

```
vect = CountVectorizer().fit(text_train)
X_train = vect.transform(text_train)
print("X_train:\n{}".format(repr(X_train)))
```

**Out[12]:**

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'>
      with 3431196 stored elements in Compressed Sparse Row format>
```

Bentuk `X_train`, representasi bag-of-words dari data pelatihan, adalah  $25.000 \times 74.849$ , menunjukkan bahwa kosakata tersebut berisi 74.849 entri. Sekali lagi, data disimpan sebagai matriks sparse SciPy. Mari kita lihat kosakata sedikit lebih detail. Cara lain untuk mengakses kosakata adalah menggunakan metode `get_feature_name` dari `vectorizer`, yang mengembalikan daftar yang nyaman di mana setiap entri sesuai dengan satu fitur:

**In[13]:**

```
feature_names = vect.get_feature_names()
print("Number of features: {}".format(len(feature_names)))
print("First 20 features:\n{}".format(feature_names[:20]))
print("Features 20010 to 20030:\n{}".format(feature_names[20010:20030]))
print("Every 2000th feature:\n{}".format(feature_names[::2000]))
```

**Out[13]:**

```

Number of features: 74849
First 20 features:
['00', '000', '000000000001', '00001', '00015', '000s', '001', '003830',
 '006', '007', '0079', '0080', '0083', '0093638', '00am', '00pm', '00s',
 '01', '01pm', '02']
Features 20010 to 20030:
['dratted', 'draub', 'draught', 'draughts', 'draughtswoman', 'draw', 'drawback',
 'drawbacks', 'drawer', 'drawers', 'drawing', 'drawings', 'drawl',
 'drawled', 'drawling', 'drawn', 'draws', 'draza', 'dre', 'drea']
Every 2000th feature:
['00', 'aesir', 'aquarian', 'barking', 'blustering', 'bête', 'chicanery',
 'condensing', 'cunning', 'detox', 'draper', 'enshrined', 'favorit', 'freezer',
 'goldman', 'hasan', 'huitieme', 'intelligible', 'kantrowitz', 'lawful',
 'maars', 'megalunged', 'mostey', 'norrland', 'padilla', 'pincher',
 'promisingly', 'receptionist', 'rivals', 'schnaas', 'shunning', 'sparse',
 'subset', 'temptations', 'treatises', 'unproven', 'walkman', 'xylophonist']

```

Seperti yang Anda lihat, mungkin sedikit mengejutkan, 10 entri pertama dalam kosakata semuanya adalah angka. Semua angka ini muncul di suatu tempat di ulasan, dan karena itu diekstraksi sebagai kata-kata. Sebagian besar dari angka-angka ini tidak memiliki makna semantik langsung—selain dari "007", yang dalam konteks film tertentu kemungkinan besar merujuk pada karakter James Bond.<sup>5</sup> Menyengkirkan makna dari "kata-kata" yang tidak bermakna adalah terkadang rumit. Melihat lebih jauh dalam kosa kata, kami menemukan kumpulan kata-kata bahasa Inggris yang dimulai dengan "dra". Anda mungkin memperhatikan bahwa untuk "draft", "kekurangan", dan "laci" baik bentuk tunggal maupun jamak terkandung dalam kosakata sebagai kata-kata yang berbeda. Kata-kata ini memiliki makna semantik yang sangat erat hubungannya, dan menghitungnya sebagai kata yang berbeda, sesuai dengan fitur yang berbeda, mungkin tidak ideal.

Sebelum kita mencoba untuk meningkatkan ekstraksi fitur kita, mari kita dapatkan ukuran kinerja kuantitatif dengan benar-benar membangun classifier. Kami memiliki label pelatihan yang disimpan di `y_train` dan representasi bag-of-words dari data pelatihan di `X_train`, sehingga kami dapat melatih pengklasifikasi pada data ini. Untuk data berdimensi tinggi dan jarang seperti ini, model linier seperti `LogisticRegression` sering kali berfungsi paling baik. Mari kita mulai dengan mengevaluasi `LogisticRegression` menggunakan validasi silang:

**In[14]:**

```

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
scores = cross_val_score(LogisticRegression(), X_train, y_train, cv=5)
print("Mean cross-validation accuracy: {:.2f}".format(np.mean(scores)))

```

**Out[14]:**

```
Mean cross-validation accuracy: 0.88
```

Kami memperoleh skor validasi silang rata-rata 88%, yang menunjukkan kinerja yang wajar untuk tugas klasifikasi biner yang seimbang. Kita tahu bahwa `LogisticRegression` memiliki parameter regularisasi, `C`, yang dapat kita atur melalui validasi silang:

**In[15]:**

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters: ", grid.best_params_)
```

**Out[15]:**

```
Best cross-validation score: 0.89
Best parameters: {'C': 0.1}
```

Kami memperoleh skor validasi silang sebesar 89% menggunakan  $C=0.1$ . Kami sekarang dapat menilai kinerja generalisasi pengaturan parameter ini pada set pengujian:

**In[16]:**

```
X_test = vect.transform(text_test)
print("{:.2f}".format(grid.score(X_test, y_test)))
```

**Out[16]:**

```
0.88
```

Sekarang, mari kita lihat apakah kita dapat meningkatkan ekstraksi kata. CountVectorizer mengekstrak token menggunakan ekspresi reguler. Secara default, ekspresi reguler yang digunakan adalah "`\b\w\w+\b`". Jika Anda tidak terbiasa dengan ekspresi reguler, ini berarti ia menemukan semua urutan karakter yang terdiri dari setidaknya dua huruf atau angka (`\w`) dan dipisahkan oleh batas kata (`\b`). Itu tidak menemukan kata-kata satu huruf, dan itu membagi kontraksi seperti "tidak" atau "bit.ly", tetapi cocok dengan "h8ter" sebagai satu kata. CountVectorizer kemudian mengubah semua kata menjadi karakter huruf kecil, sehingga "segera", "Segera", dan "segera" semuanya sesuai dengan token yang sama (dan karena itu fitur). Mekanisme sederhana ini bekerja cukup baik dalam praktiknya, tetapi seperti yang kita lihat sebelumnya, kita mendapatkan banyak fitur yang tidak informatif (seperti angka). Salah satu cara untuk mengurangi ini adalah dengan hanya menggunakan token yang muncul di setidaknya dua dokumen (atau setidaknya lima dokumen, dan seterusnya). Token yang hanya muncul dalam satu dokumen tidak mungkin muncul di set pengujian dan oleh karena itu tidak membantu. Kita dapat mengatur jumlah minimum dokumen yang diperlukan token untuk muncul dengan parameter `min_df`:

**In[17]:**

```
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
print("X_train with min_df: {}".format(repr(X_train)))
```

**Out[17]:**

```
X_train with min_df: <25000x27271 sparse matrix of type '<class 'numpy.int64'>'>
with 3354014 stored elements in Compressed Sparse Row format>
```

Dengan membutuhkan setidaknya lima penampilan dari setiap token, kita dapat menurunkan jumlah fitur menjadi 27.271, seperti yang terlihat pada keluaran sebelumnya—hanya sekitar sepertiga dari fitur asli. Mari kita lihat beberapa token lagi:

**In[18]:**

```
feature_names = vect.get_feature_names()

print("First 50 features:\n{}".format(feature_names[:50]))
print("Features 20010 to 20030:\n{}".format(feature_names[20010:20030]))
print("Every 700th feature:\n{}".format(feature_names[::700]))
```

**Out[18]:**

```
First 50 features:
['00', '000', '007', '00s', '01', '02', '03', '04', '05', '06', '07', '08',
 '09', '10', '100', '1000', '100th', '101', '102', '103', '104', '105', '107',
 '108', '10s', '10th', '11', '110', '112', '116', '117', '11th', '12', '120',
 '12th', '13', '135', '13th', '14', '140', '14th', '15', '150', '15th', '16',
 '160', '1600', '16mm', '16s', '16th']

Features 20010 to 20030:
['repentance', 'repercussions', 'repertoire', 'repetition', 'repetitions',
 'repetitious', 'repetitive', 'rephrase', 'replace', 'replaced', 'replacement',
 'replaces', 'replacing', 'replay', 'replayable', 'replayed', 'replaying',
 'replays', 'replete', 'replica']

Every 700th feature:
['00', 'affections', 'appropriately', 'barbra', 'blurbs', 'butchered',
 'cheese', 'commitment', 'courts', 'deconstructed', 'disgraceful', 'dvds',
 'eschews', 'fell', 'freezer', 'goriest', 'hauser', 'hungary', 'insinuate',
 'juggle', 'leering', 'maelstrom', 'messiah', 'music', 'occasional', 'parking',
 'pleasantville', 'pronunciation', 'recipient', 'reviews', 'sas', 'shea',
 'sneers', 'steiger', 'swastika', 'thrusting', 'tvs', 'vampyre', 'westerns']
```

Jelas ada lebih sedikit angka, dan beberapa kata yang lebih kabur atau salah ejaan tampaknya telah hilang. Mari kita lihat seberapa baik kinerja model kita dengan melakukan pencarian grid lagi:

**In[19]:**

```
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

**Out[19]:**

```
Best cross-validation score: 0.89
```

Akurasi validasi terbaik dari pencarian grid masih 89%, tidak berubah dari sebelumnya. Kami tidak meningkatkan model kami, tetapi memiliki lebih sedikit fitur untuk mempercepat pemrosesan dan membuang fitur yang tidak berguna mungkin membuat model lebih dapat diinterpretasikan.



Jika metode transformasi CountVectorizer dipanggil pada dokumen yang berisi kata-kata yang tidak terdapat dalam data pelatihan, kata-kata ini akan diabaikan karena bukan bagian dari kamus. Ini sebenarnya bukan masalah untuk klasifikasi, karena tidak mungkin mempelajari apa pun tentang kata-kata yang tidak ada dalam data pelatihan. Untuk beberapa aplikasi, seperti deteksi spam, mungkin berguna untuk menambahkan fitur yang mengkodekan berapa banyak kata yang disebut "kehabisan kosa kata" yang ada dalam dokumen tertentu. Agar ini berfungsi, Anda perlu mengatur min\_df; jika tidak, fitur ini tidak akan pernah aktif selama pelatihan.

## Stopwords

Cara lain yang bisa kita lakukan untuk menghilangkan kata-kata yang tidak informatif adalah dengan membuang kata-kata yang terlalu sering menjadi informatif. Ada dua pendekatan utama: menggunakan daftar stopword khusus bahasa, atau membuang kata-kata yang terlalu sering muncul. scikit-learn memiliki daftar stopword bahasa Inggris bawaan di modul feature\_extraction.text:

**In[20]:**

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print("Number of stop words: {}".format(len(ENGLISH_STOP_WORDS)))
print("Every 10th stopword:\n{}".format(list(ENGLISH_STOP_WORDS)[::10]))
```

**Out[20]:**

```
Number of stop words: 318
Every 10th stopword:
['above', 'elsewhere', 'into', 'well', 'rather', 'fifteen', 'had', 'enough',
 'herein', 'should', 'third', 'although', 'more', 'this', 'none', 'seemed',
 'nobody', 'seems', 'he', 'also', 'fill', 'anyone', 'anything', 'me', 'the',
 'yet', 'go', 'seeming', 'front', 'beforehand', 'forty', 'i']
```

Jelas, menghapus stopwords dalam daftar hanya dapat mengurangi jumlah fitur dengan panjang daftar-di sini, 318-tetapi mungkin menyebabkan peningkatan kinerja. Mari kita coba:

**In[21]:**

```
# Specifying stop_words="english" uses the built-in list.
# We could also augment it and pass our own.
vect = CountVectorizer(min_df=5, stop_words="english").fit(text_train)
X_train = vect.transform(text_train)
print("X_train with stop words:\n{}".format(repr(X_train)))
```

**Out[21]:**

```
X_train with stop words:
<25000x26966 sparse matrix of type '<class 'numpy.int64'>'>
with 2149958 stored elements in Compressed Sparse Row format>
```

Sekarang ada 305 (27.271–26.966) fitur yang lebih sedikit dalam kumpulan data, yang berarti bahwa sebagian besar, tetapi tidak semua, stopword muncul. Mari kita jalankan pencarian grid lagi:

**In[22]:**

```
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

**Out[22]:**

```
Best cross-validation score: 0.88
```

Performa pencarian grid sedikit menurun menggunakan stopwords—tidak cukup untuk dikhawatirkan, tetapi mengingat bahwa mengecualikan 305 fitur dari lebih dari 27.000 tidak mungkin banyak mengubah performa atau interpretasi, sepertinya tidak layak menggunakan daftar ini. Daftar tetap sangat membantu untuk kumpulan data kecil, yang mungkin tidak berisi informasi yang cukup bagi model untuk menentukan kata mana yang

merupakan stopword dari data itu sendiri. Sebagai latihan, Anda dapat mencoba pendekatan lain, membuang kata-kata yang sering muncul, dengan menyetel opsi `max_df` dari `CountVectorizer` dan melihat bagaimana pengaruhnya terhadap jumlah fitur dan kinerja.

## 7.6 RESCALING DATA DENGAN TF-IDF

Alih-alih membuang fitur yang dianggap tidak penting, pendekatan lain adalah dengan menskalakan ulang fitur berdasarkan seberapa informatif yang kami harapkan. Salah satu cara yang paling umum untuk melakukannya adalah dengan menggunakan istilah frekuensi-invers dokumen frekuensi (tf-idf) metode. Intuisi metode ini adalah untuk memberikan bobot yang tinggi pada setiap istilah yang sering muncul dalam dokumen tertentu, tetapi tidak dalam banyak dokumen dalam korpus. Jika sebuah kata sering muncul dalam dokumen tertentu, tetapi tidak dalam banyak dokumen, kemungkinan besar kata tersebut sangat deskriptif tentang isi dokumen tersebut. scikit-learn mengimplementasikan metode tf-idf dalam dua kelas: `TfidfTransformer`, yang mengambil output matriks sparse yang dihasilkan oleh `CountVectorizer` dan mengubahnya, dan `TfidfVectorizer`, yang mengambil data teks dan melakukan ekstraksi fitur bag-of-words dan transformasi tf-idf. Ada beberapa varian skema penskalaan ulang tf-idf, yang dapat Anda baca di Wikipedia. Skor tf-idf untuk kata w dalam dokumen d seperti yang diterapkan di kelas `TfidfTransformer` dan `TfidfVectorizer` diberikan oleh:

$$\text{tfidf}(w, d) = \text{tf} \log \left( \frac{N + 1}{N_w + 1} \right) + 1$$

di mana N adalah jumlah dokumen dalam set pelatihan,  $N_w$  adalah jumlah dokumen dalam set pelatihan tempat kata w muncul, dan tf (frekuensi istilah) adalah berapa kali kata w muncul di dokumen kueri d (dokumen yang ingin Anda ubah atau enkode). Kedua kelas juga menerapkan normalisasi L2 setelah menghitung representasi tf-idf; dengan kata lain, mereka mengubah skala representasi setiap dokumen untuk memiliki norma Euclidean 1. Rescaling dengan cara ini berarti bahwa panjang dokumen (jumlah kata) tidak mengubah representasi vektor.

Karena tf-idf benar-benar menggunakan properti statistik dari data pelatihan, kami akan menggunakan pipa, seperti yang dijelaskan dalam Bab 6, untuk memastikan hasil pencarian grid kami valid. Ini mengarah ke kode berikut:

**In[23]:**

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(TfidfVectorizer(min_df=5, norm=None),
                      LogisticRegression())
param_grid = {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10]}

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

**Out[23]:**

Best cross-validation score: 0.89

Seperti yang Anda lihat, ada beberapa peningkatan saat menggunakan tf-idf daripada hanya jumlah kata. Kami juga dapat memeriksa kata mana yang menurut tf-idf paling penting. Ingatlah bahwa penskalaan tf-idf dimaksudkan untuk menemukan kata-kata yang membedakan dokumen, tetapi ini adalah teknik yang murni tanpa pengawasan. Jadi, "penting" di sini tidak selalu terkait dengan label "ulasan positif" dan "ulasan negatif" yang kami minati. Pertama, kami mengekstrak TfidfVectorizer dari saluran:

**In[24]:**

```
vectorizer = grid.best_estimator_.named_steps["tfidfvectorizer"]
# transform the training dataset
X_train = vectorizer.transform(text_train)
# find maximum value for each of the features over the dataset
max_value = X_train.max(axis=0).toarray().ravel()
sorted_by_tfidf = max_value.argsort()
# get feature names
feature_names = np.array(vectorizer.get_feature_names())

print("Features with lowest tfidf:\n{}".format(
    feature_names[sorted_by_tfidf[:20]]))

print("Features with highest tfidf: \n{}".format(
    feature_names[sorted_by_tfidf[-20:]]))
```

**Out[24]:**

```
Features with lowest tfidf:
['poignant' 'disagree' 'instantly' 'importantly' 'lacked' 'occurred'
 'currently' 'altogether' 'nearby' 'undoubtedly' 'directs' 'fond' 'stinker'
 'avoided' 'emphasis' 'commented' 'disappoint' 'realizing' 'downhill'
 'inane']

Features with highest tfidf:
['coop' 'homer' 'dillinger' 'hackenstein' 'gadget' 'taker' 'macarthur'
 'vargas' 'jesse' 'basket' 'dominick' 'the' 'victor' 'bridget' 'victoria'
 'khouri' 'zizek' 'rob' 'timon' 'titanic']
```

Fitur dengan tf-idf rendah adalah fitur yang sangat umum digunakan di seluruh dokumen atau hanya digunakan sedikit, dan hanya dalam dokumen yang sangat panjang. Menariknya, banyak fitur tf-idf tinggi sebenarnya mengidentifikasi acara atau film tertentu. Istilah-istilah ini hanya muncul di ulasan untuk acara atau waralaba tertentu, tetapi cenderung sangat sering muncul di ulasan khusus ini. Ini sangat jelas, misalnya, untuk "pokemon", "smallville", dan "doodlebops", tetapi "scanner" di sini sebenarnya juga mengacu pada judul film. Kata-kata ini tidak mungkin membantu kami dalam tugas klasifikasi sentimen kami (kecuali mungkin beberapa waralaba diulas secara universal secara positif atau negatif) tetapi tentu saja mengandung banyak informasi spesifik tentang ulasan tersebut.

Kita juga dapat menemukan kata-kata yang memiliki frekuensi dokumen terbalik yang rendah—yaitu, yang sering muncul dan oleh karena itu dianggap kurang penting. Nilai frekuensi dokumen terbalik yang ditemukan pada set pelatihan disimpan dalam atribut idf\_:

**In[25]:**

```
sorted_by_idf = np.argsort(vectorizer.idf_)
print("Features with lowest idf:\n{}".format(
    feature_names[sorted_by_idf[:100]]))
```

Out[25]:

```
Features with lowest idf:  
['the', 'and', 'of', 'to', 'this', 'is', 'it', 'in', 'that', 'but', 'for', 'with',  
'was', 'as', 'on', 'movie', 'not', 'have', 'one', 'be', 'film', 'are', 'you', 'all',  
'at', 'an', 'by', 'so', 'from', 'like', 'who', 'they', 'there', 'if', 'his', 'out',  
'just', 'about', 'he', 'or', 'has', 'what', 'some', 'good', 'can', 'more', 'when',  
'time', 'up', 'very', 'even', 'only', 'no', 'would', 'my', 'see', 'really', 'story',  
'which', 'well', 'had', 'me', 'than', 'much', 'their', 'get', 'were', 'other',  
'been', 'do', 'most', 'don', 'her', 'also', 'into', 'first', 'made', 'how', 'great',  
'because', 'will', 'people', 'make', 'way', 'could', 'we', 'bad', 'after', 'any',  
'too', 'then', 'them', 'she', 'watch', 'think', 'acting', 'movies', 'seen', 'its',  
'him']
```

Seperti yang diharapkan, ini sebagian besar adalah stopword bahasa Inggris seperti "the" dan "no". Tetapi beberapa jelas spesifik domain untuk ulasan film, seperti "film", "film", "waktu", "cerita", dan sebagainya. Menariknya, "baik", "hebat", dan "buruk" juga termasuk di antara kata-kata yang paling sering dan karena itu "paling tidak relevan" menurut ukuran tf-idf, meskipun kita mungkin berharap ini menjadi sangat penting untuk sentimen kita. tugas analisis.

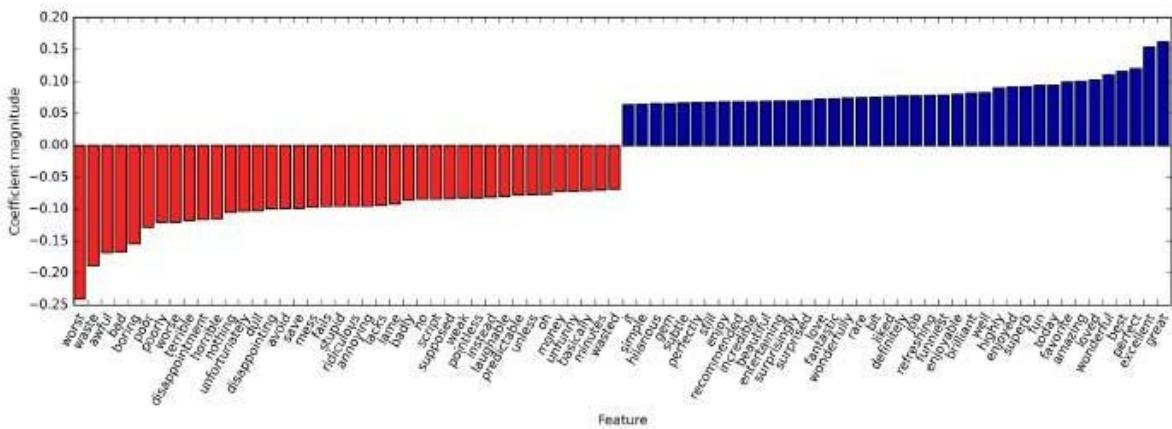
## 7.7 MENYELIDIKI KOEFISIEN MODEL

Terakhir, mari kita lihat lebih detail apa yang sebenarnya dipelajari model regresi logistik kita dari data. Karena ada begitu banyak fitur—27.271 setelah menghapus fitur yang jarang—kita jelas tidak dapat melihat semua koefisien pada saat yang bersamaan. Namun, kita dapat melihat koefisien terbesar, dan melihat kata mana yang sesuai dengan ini. Kami akan menggunakan model terakhir yang kami latih, berdasarkan fitur tf-idf.

Bagan batang berikut (Gambar 7.2) menunjukkan 25 koefisien terbesar dan 25 terkecil dari model regresi logistik, dengan batang menunjukkan ukuran masing-masing koefisien:

In[26]:

```
mglearn.tools.visualize_coefficients(  
    grid.best_estimator_.named_steps["logisticregression"].coef_,  
    feature_names, n_top_features=40)
```



**Gambar 7.2** Koefisien regresi logistik terbesar dan terkecil yang dilatih pada fitur tf-idf

Koefisien negatif di sebelah kiri termasuk kata-kata yang menurut model menunjukkan ulasan negatif, sedangkan koefisien positif di sebelah kanan termasuk kata-kata yang menurut

model menunjukkan ulasan positif. Sebagian besar istilah cukup intuitif, seperti "terburuk", "pemborosan", "kekecewaan", dan "menggelikan" menunjukkan ulasan film yang buruk, sementara "sangat baik", "luar biasa", "menyenangkan", dan "menyegarkan" menunjukkan ulasan film positif. Beberapa kata agak kurang jelas, seperti "sedikit", "pekerjaan", dan "hari ini", tetapi ini mungkin bagian dari frasa seperti "pekerjaan bagus" atau "terbaik hari ini".

## 7.8 BAG-OF-WORDS DENGAN LEBIH DARI SATU KATA (N-GRAMS)

Salah satu kelemahan utama menggunakan representasi bag-of-words adalah urutan kata benar-benar dibuang. Oleh karena itu, dua string "itu buruk, tidak baik sama sekali" dan "baik, tidak buruk sama sekali" memiliki representasi yang persis sama, meskipun maknanya dibalik. Menempatkan "tidak" di depan sebuah kata hanyalah salah satu contoh (jika ekstrim) tentang bagaimana konteks itu penting. Untungnya, ada cara untuk menangkap konteks saat menggunakan representasi bag-of-words, dengan tidak hanya mempertimbangkan jumlah token tunggal, tetapi juga jumlah pasangan atau triplet token yang muncul bersebelahan. Sepasang token dikenal sebagai bigrams, triplet token dikenal sebagai trigram, dan lebih umum urutan token dikenal sebagai n-gram. Kita dapat mengubah rentang token yang dianggap sebagai fitur dengan mengubah parameter `ngram_range` dari `CountVectorizer` atau `TfidfVectorizer`. Parameter `ngram_range` adalah tuple, terdiri dari panjang minimum dan panjang maksimum dari urutan token yang dipertimbangkan. Berikut adalah contoh data mainan yang kita gunakan sebelumnya:

**In[27]:**

```
print("bards_words:\n{}".format(bards_words))
```

**Out[27]:**

```
bards_words:
['The fool doth think he is wise,
 'but the wise man knows himself to be a fool']
```

Standarnya adalah membuat satu fitur per urutan token yang panjangnya setidaknya satu token dan paling banyak satu token, atau dengan kata lain tepat satu token (token tunggal juga disebut unigram):

**In[28]:**

```
cv = CountVectorizer(ngram_range=(1, 1)).fit(bards_words)
print("Vocabulary size: {}".format(len(cv.vocabulary_)))
print("Vocabulary:\n{}".format(cv.get_feature_names_()))
```

**Out[28]:**

```
Vocabulary size: 13
Vocabulary:
['be', 'but', 'doth', 'fool', 'he', 'himself', 'is', 'knows', 'man', 'the',
 'think', 'to', 'wise']
```

Untuk melihat hanya pada bigram—yaitu, hanya pada urutan dua token yang mengikuti satu sama lain—kita dapat menyetel `ngram_range` ke (2, 2):

**In[29]:**

```
cv = CountVectorizer(ngram_range=(2, 2)).fit(bards_words)
print("Vocabulary size: {}".format(len(cv.vocabulary_)))
print("Vocabulary:\n{}".format(cv.get_feature_names()))
```

**Out[29]:**

```
Vocabulary size: 14
Vocabulary:
['be fool', 'but the', 'doth think', 'fool doth', 'he is', 'himself to',
 'is wise', 'knows himself', 'man knows', 'the fool', 'the wise',
 'think he', 'to be', 'wise man']
```

Menggunakan urutan token yang lebih panjang biasanya menghasilkan lebih banyak fitur, dan fitur yang lebih spesifik. Tidak ada bigram umum antara dua frasa di bard\_words:

**In[30]:**

```
print("Transformed data (dense):\n{}".format(cv.transform(bards_words).toarray()))
```

**Out[30]:**

```
Transformed data (dense):
[[0 0 1 1 1 0 1 0 0 1 0 1 0 0]
 [1 1 0 0 0 1 0 1 1 0 1 0 1 1]]
```

## 7.9 TOKENISASI, STEMMING, DAN LEMMATISASI TINGKAT LANJUT

Untuk sebagian besar aplikasi, jumlah token minimum harus satu, karena kata tunggal sering kali memiliki banyak arti. Menambahkan bigram membantu dalam banyak kasus. Menambahkan urutan yang lebih panjang—hingga 5 gram—mungkin membantu juga, tetapi ini akan menyebabkan ledakan jumlah fitur dan mungkin menyebabkan overfitting, karena akan ada banyak fitur yang sangat spesifik. Pada prinsipnya, jumlah bigram bisa menjadi jumlah unigram kuadrat dan jumlah trigram bisa menjadi jumlah unigram pangkat tiga, yang mengarah ke ruang fitur yang sangat besar. Dalam praktiknya, jumlah n-gram yang lebih tinggi yang sebenarnya muncul dalam data jauh lebih kecil, karena struktur bahasa (Inggris), meskipun masih besar.

Berikut adalah tampilan menggunakan unigram, bigram, dan trigram pada bards\_words:

**In[31]:**

```
cv = CountVectorizer(ngram_range=(1, 3)).fit(bards_words)
print("Vocabulary size: {}".format(len(cv.vocabulary_)))
print("Vocabulary:\n{}".format(cv.get_feature_names()))
```

**Out[31]:**

```
Vocabulary size: 39
Vocabulary:
['be', 'be fool', 'but', 'but the', 'but the wise', 'doth', 'doth think',
 'doth think he', 'fool', 'fool doth', 'fool doth think', 'he', 'he is',
 'he is wise', 'himself', 'himself to', 'himself to be', 'is', 'is wise',
 'knows', 'knows himself', 'knows himself to', 'man', 'man knows',
 'man knows himself', 'the', 'the fool', 'the fool doth', 'the wise',
 'the wise man', 'think', 'think he', 'think he is', 'to', 'to be',
 'to be fool', 'wise', 'wise man', 'wise man knows']
```

Mari kita coba TfidfVectorizer pada data ulasan film IMDb dan temukan pengaturan terbaik rentang n-gram menggunakan pencarian grid:

**In[32]:**

```
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression())
# running the grid search takes a long time because of the
# relatively large grid and the inclusion of trigrams
param_grid = {"logisticregression__C": [0.001, 0.01, 0.1, 1, 10, 100],
              "tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (1, 3)]}

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train, y_train)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
print("Best parameters:\n{}".format(grid.best_params_))
```

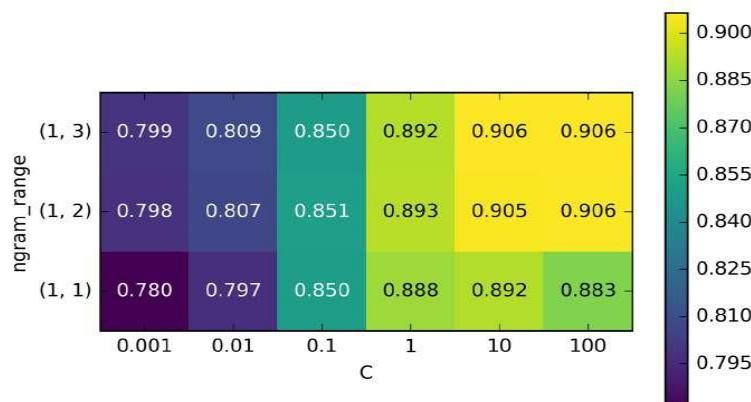
**Out[32]:**

```
Best cross-validation score: 0.91
Best parameters:
{'tfidfvectorizer__ngram_range': (1, 3), 'logisticregression__C': 100}
```

Seperti yang Anda lihat dari hasilnya, kami meningkatkan kinerja sedikit lebih dari satu persen dengan menambahkan fitur bigram dan trigram. Kita dapat memvisualisasikan akurasi validasi silang sebagai fungsi dari ngram\_range dan parameter C sebagai peta panas, seperti yang kita lakukan di Bab 5 (lihat Gambar 7.3):

**In[33]:**

```
# extract scores from grid_search
scores = grid.cv_results_['mean_test_score'].reshape(-1, 3).T
# visualize heat map
heatmap = mglearn.tools.heatmap(
    scores, xlabel="C", ylabel="ngram_range", cmap="viridis", fmt=".3f",
    xticklabels=param_grid['logisticregression__C'],
    yticklabels=param_grid['tfidfvectorizer__ngram_range'])
plt.colorbar(heatmap)
```



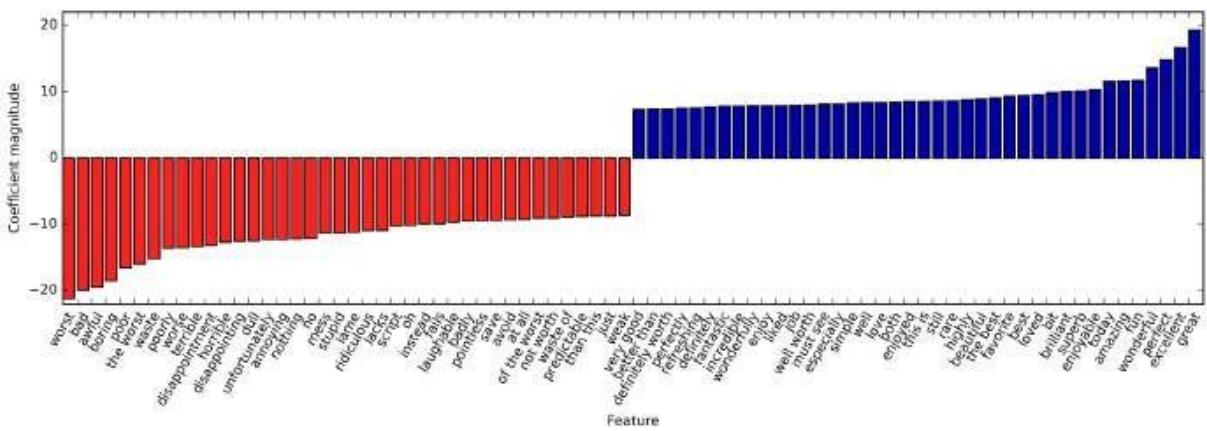
**Gambar 7.3** Visualisasi peta panas akurasi validasi rata-rata sebagai fungsi dari parameter ngram\_range dan C

Dari peta panas kita dapat melihat bahwa menggunakan bigram meningkatkan kinerja sedikit, sementara menambahkan trigram hanya memberikan manfaat yang sangat kecil dalam hal akurasi. Untuk memahami lebih baik bagaimana model ditingkatkan, kita dapat

memvisualisasikan koefisien penting untuk model terbaik, yang mencakup unigram, bigram, dan trigram (lihat Gambar 7.4):

**In[34]:**

```
# extract feature names and coefficients
vect = grid.best_estimator_.named_steps['tfidfvectorizer']
feature_names = np.array(vect.get_feature_names())
coef = grid.best_estimator_.named_steps['logisticregression'].coef_
mglearn.tools.visualize_coefficients(coef, feature_names, n_top_features=40)
```



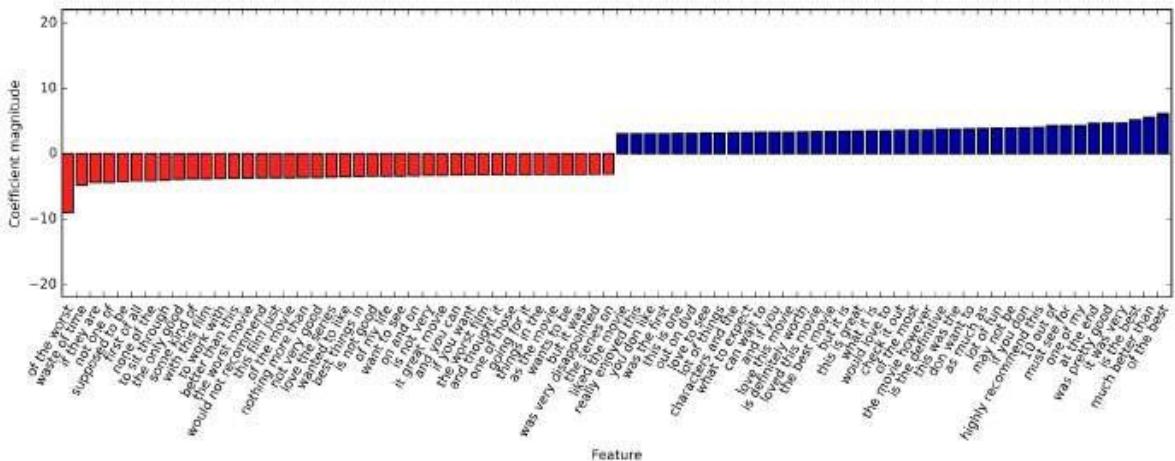
**Gambar 7.4** Fitur terpenting saat menggunakan unigram, bigram, dan trigram dengan tf-idf rescaling

Ada fitur yang sangat menarik yang mengandung kata "layak" yang tidak ada dalam model unigram: "tidak layak" menunjukkan ulasan negatif, sementara "sangat berharga" dan "sangat berharga" menunjukkan ulasan positif. Ini adalah contoh utama dari konteks yang mempengaruhi arti kata "berharga".

Selanjutnya, kita hanya akan memvisualisasikan trigram, untuk memberikan wawasan lebih lanjut tentang mengapa fitur ini berguna. Banyak bigram dan trigram yang berguna terdiri dari kata-kata umum yang tidak akan informatif dengan sendirinya, seperti dalam frasa "tidak satu pun dari", "satu-satunya yang baik", "terus dan terus", "ini adalah satu", "dari paling", dan seterusnya. Namun, dampak dari fitur ini cukup terbatas dibandingkan dengan pentingnya fitur unigram, seperti yang Anda lihat pada Gambar 7.5:

**In[35]:**

```
# find 3-gram features
mask = np.array([len(feature.split(" ")) for feature in feature_names]) == 3
# visualize only 3-gram features
mglearn.tools.visualize_coefficients(coef.ravel()[mask],
                                     feature_names[mask], n_top_features=40)
```



**Gambar 7.8** Tokenisasi, Stemming, dan Lemmatization Tingkat Lanjut

Seperti disebutkan sebelumnya, ekstraksi fitur di CountVectorizer dan TfidfVectorizer relatif sederhana, dan metode yang jauh lebih rumit dimungkinkan. Satu langkah tertentu yang sering ditingkatkan dalam aplikasi pemrosesan teks yang lebih canggih adalah langkah pertama dalam model *bag-of-words*: *tokenization*. Langkah ini mendefinisikan apa yang merupakan kata untuk tujuan ekstraksi fitur.

Kita telah melihat sebelumnya bahwa kosakata sering mengandung versi tunggal dan jamak dari beberapa kata, seperti dalam "kekurangan" dan "kekurangan", "laci" dan "laci", dan "gambar" dan "gambar". Untuk keperluan model *bag-of-words*, semantik "kelemahan" dan "kekurangan" begitu dekat sehingga membedakan mereka hanya akan meningkatkan overfitting, dan tidak memungkinkan model untuk sepenuhnya mengeksplorasi data pelatihan. Demikian pula, kami menemukan kosakata termasuk kata-kata seperti "mengganti", "mengganti", "mengganti", "mengganti", dan "mengganti", yang merupakan bentuk kata kerja yang berbeda dan kata benda yang berkaitan dengan kata kerja "mengganti". Sama halnya dengan memiliki bentuk tunggal dan jamak dari kata benda, memperlakukan bentuk kata kerja yang berbeda dan kata-kata terkait sebagai tanda yang berbeda adalah tidak menguntungkan untuk membangun model yang dapat digeneralisasi dengan baik.

Masalah ini dapat diatasi dengan merepresentasikan setiap kata menggunakan batang kata, yang melibatkan mengidentifikasi (atau menggabungkan) semua kata yang memiliki batang kata yang sama. Jika ini dilakukan dengan menggunakan heuristik berbasis aturan, seperti menjatuhkan sufiks umum, biasanya disebut sebagai stemming. Jika sebagai gantinya kamus bentuk kata yang dikenal digunakan (sistem eksplisit dan diverifikasi manusia), dan peran kata dalam kalimat diperhitungkan, prosesnya disebut lemmatisasi dan bentuk kata standar disebut untuk sebagai lemma. Kedua metode pemrosesan, lemmatization dan stemming, adalah bentuk normalisasi yang mencoba mengekstrak beberapa bentuk normal dari sebuah kata. Kasus normalisasi lain yang menarik adalah koreksi ejaan, yang dapat membantu dalam praktik tetapi berada di luar cakupan buku ini.

Untuk mendapatkan pemahaman yang lebih baik tentang normalisasi, mari kita bandingkan metode untuk stemming — the Porter stemmer, kumpulan heuristik yang banyak digunakan (di sini diimpor dari paket nltk) — dengan lemmatisasi seperti yang diterapkan dalam paket spacy:

**In[36]:**

```
import spacy
import nltk

# load spacy's English-language models
en_nlp = spacy.load('en')
# instantiate nltk's Porter stemmer
stemmer = nltk.stem.PorterStemmer()

# define function to compare lemmatization in spacy with stemming in nltk
def compare_normalization(doc):
    # tokenize document in spacy
    doc_spacy = en_nlp(doc)
    # print lemmas found by spacy
    print("Lemmatization:")
    print([token.lemma_ for token in doc_spacy])
    # print tokens found by Porter stemmer
    print("Stemming:")
    print([stemmer.stem(token.norm_.lower()) for token in doc_spacy])
```

Kami akan membandingkan lemmatization dan Porter stemmer pada kalimat yang dirancang untuk menunjukkan beberapa perbedaan:

**In[37]:**

```
compare_normalization(u"Our meeting today was worse than yesterday, "
                      "I'm scared of meeting the clients tomorrow.")
```

**Out[37]:**

```
Lemmatization:
['our', 'meeting', 'today', 'be', 'bad', 'than', 'yesterday', ',', 'i', 'be',
 'scared', 'of', 'meet', 'the', 'client', 'tomorrow', '.']
Stemming:
['our', 'meet', 'today', 'wa', 'wors', 'than', 'yesterday', ',', 'i', "'m",
 'scare', 'of', 'meet', 'the', 'client', 'tomorrow', '.']
```

Stemming selalu dibatasi untuk memangkas kata menjadi batang, jadi "was" menjadi "wa", sedangkan lemmatization dapat mengambil bentuk kata kerja dasar yang benar, "be". Demikian pula, lemmatisasi dapat menormalkan "lebih buruk" menjadi "buruk", sedangkan stemming menghasilkan "buruk". Perbedaan utama lainnya adalah bahwa stemming mengurangi kedua kemunculan "bertemu" menjadi "bertemu". Dengan menggunakan lemmatization, kemunculan pertama "bertemu" dikenali sebagai kata benda dan dibiarkan apa adanya, sedangkan kemunculan kedua dikenali sebagai kata kerja dan direduksi menjadi "bertemu". Secara umum, lemmatisasi adalah proses yang jauh lebih terlibat daripada stemming, tetapi biasanya menghasilkan hasil yang lebih baik daripada stemming ketika digunakan untuk menormalkan token untuk pembelajaran mesin.

Sementara scikit-learn tidak mengimplementasikan kedua bentuk normalisasi, CountVectorizer memungkinkan menentukan tokenizer Anda sendiri untuk mengonversi setiap dokumen menjadi daftar token menggunakan parameter tokenizer. Kita dapat

menggunakan lemmatization dari spacy untuk membuat callable yang akan mengambil string dan menghasilkan daftar lemma:

**In[38]:**

```
# Technicality: we want to use the regexp-based tokenizer
# that is used by CountVectorizer and only use the lemmatization
# from spacy. To this end, we replace en_nlp.tokenizer (the spacy tokenizer)
# with the regexp-based tokenization.
import re
# regexp used in CountVectorizer
regexp = re.compile('(?u)\\b\\w\\w+\\b')

# load spacy language model and save old tokenizer
en_nlp = spacy.load('en')
old_tokenizer = en_nlp.tokenizer
# replace the tokenizer with the preceding regexp
en_nlp.tokenizer = lambda string: old_tokenizer.tokens_from_list(
    regexp.findall(string))

# create a custom tokenizer using the spacy document processing pipeline
# (now using our own tokenizer)
def custom_tokenizer(document):
    doc_spacy = en_nlp(document, entity=False, parse=False)
    return [token.lemma_ for token in doc_spacy]

# define a count vectorizer with the custom tokenizer
lemma_vect = CountVectorizer(tokenizer=custom_tokenizer, min_df=5)
```

Mari kita ubah data dan periksa ukuran kosakata:

**In[39]:**

```
# transform text_train using CountVectorizer with lemmatization
X_train_lemma = lemma_vect.fit_transform(text_train)
print("X_train_lemma.shape: {}".format(X_train_lemma.shape))

# standard CountVectorizer for reference
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
print("X_train.shape: {}".format(X_train.shape))
```

**Out[39]:**

```
X_train_lemma.shape: (25000, 21596)
X_train.shape: (25000, 27271)
```

Seperti yang Anda lihat dari output, lemmatization mengurangi jumlah fitur dari 27.271 (dengan pemrosesan CountVectorizer standar) menjadi 21.596. Lemmatisasi dapat dilihat sebagai semacam regularisasi, karena menggabungkan fitur-fitur tertentu. Oleh karena itu, kami mengharapkan lemmatisasi untuk meningkatkan kinerja paling banyak ketika kumpulan data kecil. Untuk mengilustrasikan bagaimana lemmatisasi dapat membantu, kita akan menggunakan StratifiedShuffleSplit untuk validasi silang, menggunakan hanya 1% dari data sebagai data pelatihan dan sisanya sebagai data uji:

**In[40]:**

```
# build a grid search using only 1% of the data as the training set
from sklearn.model_selection import StratifiedShuffleSplit

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
cv = StratifiedShuffleSplit(n_iter=5, test_size=0.99,
                           train_size=0.01, random_state=0)
grid = GridSearchCV(LogisticRegression(), param_grid, cv=cv)
# perform grid search with standard CountVectorizer
grid.fit(X_train, y_train)
print("Best cross-validation score "
      "(standard CountVectorizer): {:.3f}".format(grid.best_score_))
# perform grid search with lemmatization
grid.fit(X_train_lemma, y_train)
print("Best cross-validation score "
      "(lemmatization): {:.3f}".format(grid.best_score_))
```

**Out[40]:**

```
Best cross-validation score (standard CountVectorizer): 0.721
Best cross-validation score (lemmatization): 0.731
```

Dalam hal ini, lemmatisasi memberikan sedikit peningkatan kinerja. Seperti banyak teknik ekstraksi fitur yang berbeda, hasilnya bervariasi tergantung pada kumpulan data. Lemmatisasi dan stemming terkadang dapat membantu dalam membangun model yang lebih baik (atau setidaknya lebih ringkas), jadi kami sarankan Anda mencoba teknik ini saat mencoba memeras sedikit performa terakhir pada tugas tertentu.

## 7.10 PEMODELAN TOPIK DAN PENGELOMPOKAN DOKUMEN

Salah satu teknik tertentu yang sering diterapkan pada data teks adalah pemodelan topik, yang merupakan istilah umum yang menjelaskan tugas menetapkan setiap dokumen ke satu atau beberapa topik, biasanya tanpa pengawasan. Contoh yang baik untuk ini adalah data berita, yang dapat dikategorikan ke dalam topik seperti "politik", "olahraga", "keuangan", dan seterusnya. Jika setiap dokumen diberi satu topik, ini adalah tugas mengelompokkan dokumen, seperti yang dibahas dalam Bab 3. Jika setiap dokumen dapat memiliki lebih dari satu topik, tugas tersebut berkaitan dengan metode dekomposisi dari Bab 3. Setiap komponen yang kita belajar kemudian sesuai dengan satu topik, dan koefisien komponen dalam representasi dokumen memberi tahu kita seberapa kuat keterkaitan dokumen itu dengan topik tertentu. Seringkali, ketika orang berbicara tentang pemodelan topik, mereka mengacu pada satu metode dekomposisi tertentu yang disebut Latent Dirichlet Allocation (sering disingkat LDA).

## 7.11 ALOKASI DIRICHLET LATEN

Secara intuitif, model LDA mencoba menemukan kelompok kata (topik) yang sering muncul bersamaan. LDA juga mensyaratkan bahwa setiap dokumen dapat dipahami sebagai "campuran" dari subset topik. Penting untuk dipahami bahwa untuk model pembelajaran mesin, "topik" mungkin tidak seperti yang biasanya kita sebut topik dalam percakapan sehari-hari, tetapi lebih menyerupai komponen yang diekstraksi oleh PCA atau NMF (yang telah kita bahas di Bab 3), yang mungkin atau mungkin tidak memiliki arti semantik. Bahkan jika ada arti semantik untuk "topik" LDA, itu mungkin bukan sesuatu yang biasanya kita sebut topik.

Kembali ke contoh artikel berita, kita mungkin memiliki kumpulan artikel tentang olahraga, politik, dan keuangan, yang ditulis oleh dua penulis tertentu.

Dalam artikel politik, kita mungkin berharap melihat kata-kata seperti "gubernur", "suara", "partai", dll., sedangkan dalam artikel olahraga kita mungkin mengharapkan kata-kata seperti "tim", "skor", dan "musim". Kata-kata di masing-masing grup ini kemungkinan akan muncul bersamaan, sementara kemungkinan kecil, misalnya, "tim" dan "gubernur" akan muncul bersamaan. Namun, ini bukan satu-satunya kelompok kata yang mungkin kita harapkan muncul bersama. Kedua reporter mungkin lebih menyukai frasa atau pilihan kata yang berbeda. Mungkin salah satu dari mereka suka menggunakan kata "demarcate" dan ada yang suka dengan kata "polarize". "Topik" lainnya kemudian akan menjadi "kata-kata yang sering digunakan oleh reporter A" dan "kata-kata yang sering digunakan oleh reporter B", meskipun ini bukan topik dalam arti kata yang biasa.

Mari terapkan LDA ke kumpulan data ulasan film untuk melihat cara kerjanya dalam praktik. Untuk model dokumen teks tanpa pengawasan, seringkali baik untuk menghapus kata-kata yang sangat umum, karena mungkin mendominasi analisis. Kami akan menghapus kata yang muncul di setidaknya 20 persen dokumen, dan kami akan membatasi model bag-of-words hingga 10.000 kata yang paling umum setelah menghapus 20 persen teratas:

**In[41]:**

```
vect = CountVectorizer(max_features=10000, max_df=.15)
X = vect.fit_transform(text_train)
```

Kami akan mempelajari model topik dengan 10 topik, yang cukup sedikit sehingga kami dapat melihat semuanya. Sama halnya dengan komponen di NMF, topik tidak memiliki urutan yang melekat, dan mengubah jumlah topik akan mengubah semua topik. Kami akan menggunakan metode pembelajaran "batch", yang agak lebih lambat daripada default ("online") tetapi biasanya memberikan hasil yang lebih baik, dan meningkatkan "max\_iter", yang juga dapat menghasilkan model yang lebih baik:

**In[42]:**

```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_topics=10, learning_method="batch",
                                 max_iter=25, random_state=0)
# We build the model and transform the data in one step
# Computing transform takes some time,
# and we can save time by doing both at once
document_topics = lda.fit_transform(X)
```

Seperti metode dekomposisi yang kita lihat di Bab 3, LatentDirichletAllocation memiliki atribut component\_ yang menyimpan seberapa penting setiap kata untuk setiap topik. Ukuran component\_ adalah (n\_topics, n\_words):

In[43]:

```
lda.components_.shape
```

Out[43]:

```
(10, 10000)
```

Untuk memahami lebih baik apa arti topik yang berbeda, kita akan melihat kata-kata yang paling penting untuk masing-masing topik. Fungsi print\_topics menyediakan format yang bagus untuk fitur-fitur ini:

In[44]:

```
# For each topic (a row in the components_), sort the features (ascending)
# Invert rows with [ :, ::-1] to make sorting descending
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
# Get the feature names from the vectorizer
feature_names = np.array(vect.get_feature_names())
```

In[45]:

```
# Print out the 10 topics:
mglearn.tools.print_topics(topics=range(10), feature_names=feature_names,
                           sorting=sorting, topics_per_chunk=5, n_words=10)
```

Out[45]:

topic 0	topic 1	topic 2	topic 3	topic 4
-----	-----	-----	-----	-----
between	war	funny	show	didn
young	world	worst	series	saw
family	us	comedy	episode	am
real	our	thing	tv	thought
performance	american	guy	episodes	years
beautiful	documentary	re	shows	book
work	history	stupid	season	watched
each	new	actually	new	now
both	own	nothing	television	dvd
director	point	want	years	got
topic 5	topic 6	topic 7	topic 8	topic 9
-----	-----	-----	-----	-----
horror	kids	cast	performance	house
action	action	role	role	woman
effects	animation	john	john	gets
budget	game	version	actor	killer
nothing	fun	novel	oscar	girl
original	disney	both	cast	wife
director	children	director	plays	horror
minutes	10	played	jack	young
pretty	kid	performance	joe	goes
doesn	old	mr	performances	around

Dilihat dari kata-katanya yang penting, topik 1 sepertinya tentang film sejarah dan perang, topik 2 mungkin tentang komedi yang buruk, topik 3 mungkin tentang serial TV. Topik 4 tampaknya menangkap beberapa kata yang sangat umum, sementara topik 6 tampaknya tentang film anak-anak dan topik 8 tampaknya menangkap ulasan terkait penghargaan. Dengan hanya menggunakan 10 topik, masing-masing topik harus sangat luas, sehingga mereka dapat bersama-sama mencakup semua jenis ulasan yang berbeda dalam kumpulan data kami.

Selanjutnya, kita akan mempelajari model lain, kali ini dengan 100 topik. Menggunakan lebih banyak topik membuat analisis menjadi lebih sulit, tetapi membuatnya lebih mungkin bahwa topik dapat mengkhususkan diri pada subkumpulan data yang menarik:

In[46]:

```
lda100 = LatentDirichletAllocation(n_topics=100, learning_method="batch",
                                    max_iter=25, random_state=0)
document_topics100 = lda100.fit_transform(X)
```

Melihat semua 100 topik akan sedikit berlebihan, jadi kami memilih beberapa topik yang menarik dan representatif:

In[47]:

```
topics = np.array([7, 16, 24, 25, 28, 36, 37, 45, 51, 53, 54, 63, 89, 97])
sorting = np.argsort(lda100.components_, axis=1)[:, ::-1]
feature_names = np.array(vect.get_feature_names())
mglearn.tools.print_topics(topics=topics, feature_names=feature_names,
                           sorting=sorting, topics_per_chunk=7, n_words=20)
```

Out[48]:

topic 7	topic 16	topic 24	topic 25	topic 28
thriller	worst	german	car	beautiful
suspense	awful	hitler	gets	young
horror	boring	nazi	guy	old
atmosphere	horrible	midnight	around	romantic
mystery	stupid	joe	down	between
house	thing	germany	kill	romance
director	terrible	years	goes	wonderful
quite	script	history	killed	heart
bit	nothing	new	going	feel
de	worse	modesty	house	year
performances	waste	cowboy	away	each
dark	pretty	jewish	head	french
twist	minutes	past	take	sweet
hitchcock	didn	kirk	another	boy
tension	actors	young	getting	loved
interesting	actually	spanish	doesn	girl
mysterious	re	enterprise	now	relationship
murder	supposed	von	night	saw
ending	mean	nazis	right	both
creepy	want	spock	woman	simple
topic 36	topic 37	topic 41	topic 45	topic 51
performance	excellent	war	music	earth
role	highly	american	song	space
actor	amazing	world	songs	planet
cast	wonderful	soldiers	rock	superman
play	truly	military	band	alien
actors	superb	army	soundtrack	world
performances	actors	tarzan	singing	evil
played	brilliant	soldier	voice	humans
supporting	recommend	america	singer	aliens
director	quite	country	sing	human
oscar	performance	americans	musical	creatures
roles	performances	during	roll	miike
actress	perfect	men	fan	monsters
excellent	drama	us	metal	apes
screen	without	government	concert	clark
plays	beautiful	jungle	playing	burton
award	human	vietnam	hear	tim
work	moving	ii	fans	outer
playing	world	political	prince	men
gives	recommended	against	especially	moon

topic 53	topic 54	topic 63	topic 89	topic 97
scott	money	funny	dead	didn
gary	budget	comedy	zombie	thought
streisand	actors	laugh	gore	wasn
star	low	jokes	zombies	ending
hart	worst	humor	blood	minutes
lundgren	waste	hilarious	horror	got
dolph	10	laughs	flesh	felt
career	give	fun	minutes	part
sabrina	want	re	body	going
role	nothing	funniest	living	seemed
temple	terrible	laughing	eating	bit
phantom	crap	joke	flick	found
judy	must	few	budget	though
melissa	reviews	moments	head	nothing
zorro	imdb	guy	gory	lot
gets	director	unfunny	evil	saw
barbra	thing	times	shot	long
cast	believe	laughed	low	interesting
short	am	comedies	fulci	few
serial	actually	isn	re	half

Topik yang kami rangkum kali ini tampaknya lebih spesifik, meskipun banyak yang sulit untuk ditafsirkan. Topik 7 sepertinya tentang film horor dan thriller; topik 16 dan 54 tampaknya mendapatkan ulasan buruk, sedangkan topik 63 sebagian besar tampaknya mendapatkan ulasan positif tentang komedi. Jika kita ingin membuat kesimpulan lebih lanjut menggunakan topik yang ditemukan, kita harus mengkonfirmasi intuisi yang kita peroleh dari melihat kata-kata berperingkat tertinggi untuk setiap topik dengan melihat dokumen yang ditugaskan untuk topik ini. Misalnya, topik 45 sepertinya tentang musik. Mari kita periksa jenis ulasan mana yang ditetapkan untuk topik ini:

In[49]:

```
# sort by weight of "music" topic 45
music = np.argsort(document_topics100[:, 45])[::-1]
# print the five documents where the topic is most important
for i in music[:10]:
    # pshow first two sentences
    print(b"\n".join(text_train[i].split(b"\n")[:2]) + b"\n")
```

Out[49]:

```
b'I love this movie and never get tired of watching. The music in it is great.\n'
b'I enjoyed Still Crazy more than any film I have seen in years. A successful
band from the 70's decide to give it another try.\n"
b'Hollywood Hotel was the last movie musical that Busby Berkeley directed for
Warner Bros. His directing style had changed or evolved to the point that
this film does not contain his signature overhead shots or huge production
numbers with thousands of extras.\n'
b"What happens to washed up rock-n-roll stars in the late 1990's?
They launch a comeback / reunion tour. At least, that's what the members of
Strange Fruit. a (fictional) 70's stadium rock group do.\n"
b'As a big-time Prince fan of the last three to four years, I really can't
believe I've only just got round to watching "Purple Rain". The brand new
2-disc anniversary Special Edition led me to buy it.\n'
b"This film is worth seeing alone for Jared Harris' outstanding portrayal
of John Lennon. It doesn't matter that Harris doesn't exactly resemble
Lennon; his mannerisms, expressions, posture, accent and attitude are
pure Lennon.\n"
```

```

b"The funky, yet strictly second-tier British glam-rock band Strange Fruit
breaks up at the end of the wild'n'wacky excess-ridden 70's. The individual
band members go their separate ways and uncomfortably settle into lackluster
middle age in the dull and uneventful 90's: morose keyboardist Stephen Rea
winds up penniless and down on his luck, vain, neurotic, pretentious lead
singer Bill Nighy tries (and fails) to pursue a floundering solo career,
paranoid drummer Timothy Spall resides in obscurity on a remote farm so he
can avoid paying a hefty back taxes debt, and surly bass player Jimmy Nail
installs roofs for a living.\n"
b"I just finished reading a book on Anita Loos' work and the photo in TCM
Magazine of MacDonald in her angel costume looked great (impressive wings),
so I thought I'd watch this movie. I'd never heard of the film before, so I
had no preconceived notions about it whatsoever.\n"
b'I love this movie!!! Purple Rain came out the year I was born and it has had
my heart since I can remember. Prince is so tight in this movie.\n'
b"This movie is sort of a Carrie meets Heavy Metal. It's about a highschool
guy who gets picked on alot and he totally gets revenge with the help of a
Heavy Metal ghost.\n"

```

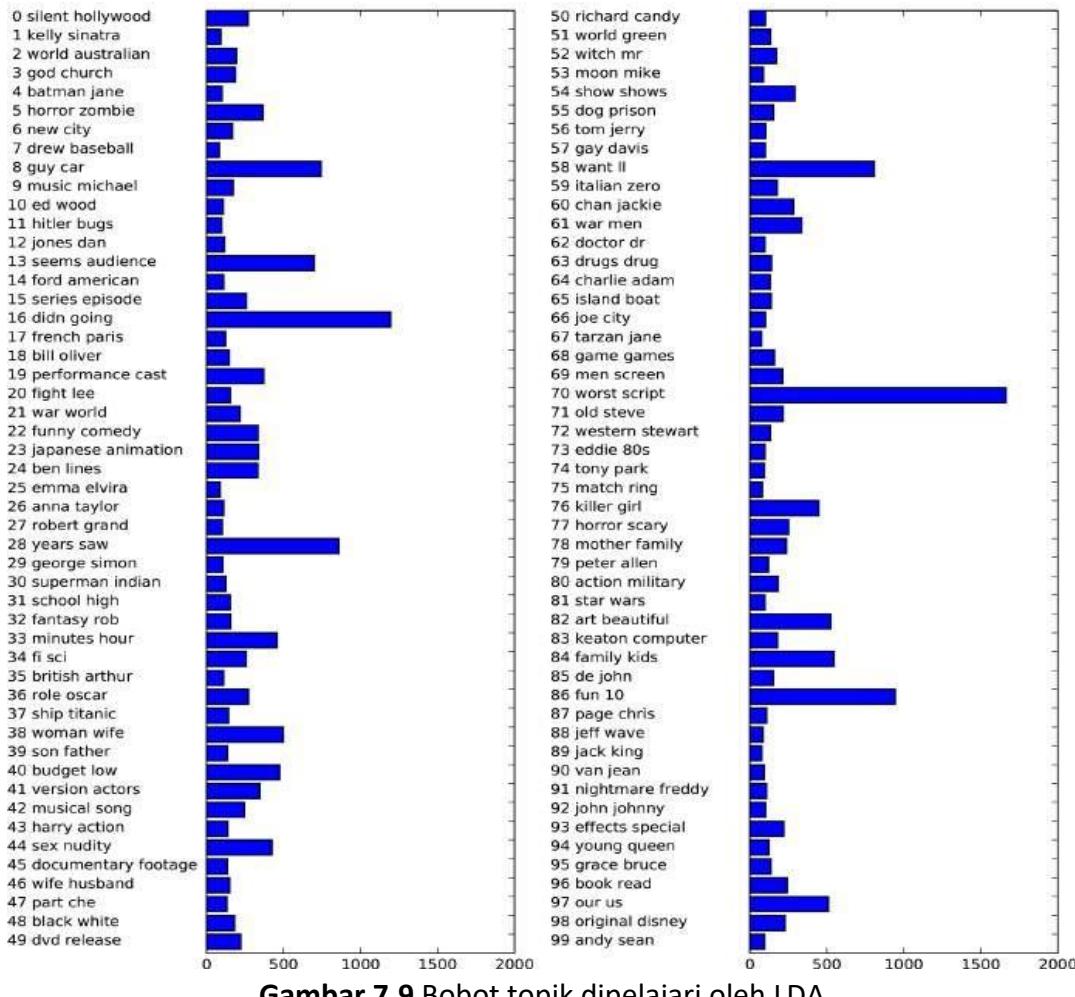
Seperti yang bisa kita lihat, topik ini mencakup berbagai macam tinjauan yang berpusat pada musik, mulai dari musical, film biografi, hingga genre yang sulit ditentukan dalam tinjauan terakhir. Cara lain yang menarik untuk memeriksa topik adalah dengan melihat seberapa besar bobot setiap topik secara keseluruhan, dengan menjumlahkan document\_topics dari semua ulasan. Kami menamai setiap topik dengan dua kata yang paling umum. Gambar 7.9 menunjukkan bobot topik yang dipelajari:

**In[50]:**

```

fig, ax = plt.subplots(1, 2, figsize=(10, 10))
topic_names = ["{:>2} ".format(i) + " ".join(words)
               for i, words in enumerate(feature_names[sorting[:, :2]])]
# two column bar chart:
for col in [0, 1]:
    start = col * 50
    end = (col + 1) * 50
    ax[col].barh(np.arange(50), np.sum(document_topics100, axis=0)[start:end])
    ax[col].set_yticks(np.arange(50))
    ax[col].set_yticklabels(topic_names[start:end], ha="left", va="top")
    ax[col].invert_yaxis()
    ax[col].set_xlim(0, 2000)
    yax = ax[col].get_yaxis()
    yax.set_tick_params(pad=130)
plt.tight_layout()

```



Gambar 7.9 Bobot topik dipelajari oleh LDA

Topik yang paling penting adalah 97, yang tampaknya sebagian besar terdiri dari stopword, mungkin dengan sedikit arah negatif; topik 16, yang jelas tentang ulasan buruk; diikuti oleh beberapa topik khusus genre dan 36 dan 37, keduanya tampaknya mengandung kata-kata puji.

Sepertinya LDA kebanyakan menemukan dua jenis topik, genre-specific dan rating-specific, selain beberapa topik yang lebih tidak spesifik. Ini adalah penemuan yang menarik, karena sebagian besar ulasan terdiri dari beberapa komentar khusus film dan beberapa komentar yang membenarkan atau menekankan peringkat.

Model topik seperti LDA adalah metode yang menarik untuk memahami kumpulan teks besar tanpa adanya label—atau, seperti di sini, bahkan jika label tersedia. Algoritme LDA diacak, dan mengubah parameter random\_state dapat menghasilkan hasil yang sangat berbeda. Meskipun mengidentifikasi topik dapat membantu, setiap kesimpulan yang Anda ambil dari model tanpa pengawasan harus diambil dengan sebutir garam, dan kami merekomendasikan untuk memverifikasi intuisi Anda dengan melihat dokumen dalam topik tertentu. Topik yang dihasilkan oleh metode LDA.transform terkadang juga dapat digunakan sebagai representasi ringkas untuk pembelajaran terawasi. Ini sangat membantu ketika beberapa contoh pelatihan tersedia.

## 7.12 RINGKASAN DAN PANDANGAN

Dalam bab ini kita berbicara tentang dasar-dasar pemrosesan teks, juga dikenal sebagai pemrosesan bahasa alami (NLP), dengan contoh aplikasi yang mengklasifikasikan ulasan film. Alat yang dibahas di sini harus berfungsi sebagai titik awal yang bagus ketika mencoba memproses data teks. Khususnya untuk tugas klasifikasi teks seperti deteksi spam dan penipuan atau analisis sentimen, representasi bag-of-words memberikan solusi yang sederhana dan kuat. Seperti yang sering terjadi dalam pembelajaran mesin, representasi data adalah kunci dalam aplikasi NLP, dan memeriksa token dan n-gram yang diekstraksi dapat memberikan wawasan yang kuat ke dalam proses pemodelan. Dalam aplikasi pemrosesan teks, seringkali mungkin untuk mengintrospeksi model dengan cara yang berarti, seperti yang kita lihat dalam bab ini, untuk tugas yang diawasi dan tidak diawasi. Anda harus memanfaatkan sepenuhnya kemampuan ini saat menggunakan metode berbasis NLP dalam praktik.

Pemrosesan bahasa dan teks alami adalah bidang penelitian yang luas, dan membahas detail metode lanjutan jauh di luar cakupan buku ini. Jika Anda ingin mempelajari lebih lanjut, kami merekomendasikan buku O'Reilly Natural Language Processing with Python oleh Steven Bird, Ewan Klein, dan Edward Loper, yang memberikan ikhtisar NLP bersama dengan pengantar paket nltk Python untuk NLP. Buku hebat dan lebih konseptual lainnya adalah referensi standar Pengantar Pengambilan Informasi oleh Christopher Manning, Prabhakar Raghavan, dan Hinrich Schütze, yang menjelaskan algoritme dasar dalam pengambilan informasi, NLP, dan pembelajaran mesin.

Kedua buku tersebut memiliki versi online yang dapat diakses secara gratis. Seperti yang telah kita bahas sebelumnya, kelas CountVectorizer dan TfidfVectorizer hanya mengimplementasikan metode pemrosesan teks yang relatif sederhana. Untuk metode pemrosesan teks yang lebih maju, kami merekomendasikan paket Python spacy (paket yang relatif baru tetapi sangat efisien dan dirancang dengan baik), nltk (library yang sangat mapan dan lengkap tetapi agak ketinggalan zaman), dan gensim (paket NLP dengan penekanan pada pemodelan topik).

Ada beberapa perkembangan baru yang sangat menarik dalam pemrosesan teks dalam beberapa tahun terakhir, yang berada di luar cakupan buku ini dan berhubungan dengan jaringan saraf. Yang pertama adalah penggunaan representasi vektor kontinu, juga dikenal sebagai vektor kata atau representasi kata terdistribusi, seperti yang diterapkan di perpustakaan word2vec. Makalah asli “Representasi Terdistribusi Kata dan Frasa dan Komposisinya” oleh Thomas Mikolov et al. adalah pengantar yang bagus untuk subjek. Spacy dan gensim menyediakan fungsionalitas untuk teknik yang dibahas dalam makalah ini dan tindak lanjutnya.

Arah lain dalam NLP yang telah mengambil momentum dalam beberapa tahun terakhir adalah penggunaan jaringan saraf berulang (RNNs) untuk pemrosesan teks. RNN adalah jenis jaringan saraf yang sangat kuat yang dapat menghasilkan output berupa teks lagi, berbeda dengan model klasifikasi yang hanya dapat menetapkan label kelas. Kemampuan untuk menghasilkan teks sebagai output membuat RNN cocok untuk terjemahan dan ringkasan otomatis. Pengantar topik ini dapat ditemukan di makalah yang relatif teknis “Urutan

Pembelajaran dengan Jaringan Syaraf” oleh Ilya Sutskever, Oriol Vinyals, dan Quoc Le. Tutorial yang lebih praktis menggunakan framework tensorflow dapat ditemukan di situs web TensorFlow.

## BAB 8

### PENUTUP

Anda sekarang tahu bagaimana menerapkan algoritme pembelajaran mesin yang penting untuk pembelajaran terawasi dan tanpa pengawasan, yang memungkinkan Anda memecahkan berbagai masalah pembelajaran mesin. Sebelum kami meninggalkan Anda untuk menjelajahi semua kemungkinan yang ditawarkan pembelajaran mesin, kami ingin memberi Anda beberapa saran terakhir, mengarahkan Anda ke beberapa sumber daya tambahan, dan memberi Anda saran tentang bagaimana Anda dapat lebih meningkatkan pembelajaran mesin dan keterampilan ilmu data Anda.

#### **8.1 MASALAH PEMBELAJARAN MESIN**

Dengan semua metode hebat yang kami perkenalkan dalam buku ini sekarang di ujung jari Anda, mungkin Anda tergoda untuk terjun dan mulai memecahkan masalah terkait data Anda hanya dengan menjalankan algoritme favorit Anda. Namun, ini biasanya bukan cara yang baik untuk memulai analisis Anda. Algoritma pembelajaran mesin biasanya hanya sebagian kecil dari analisis data yang lebih besar dan proses pengambilan keputusan. Untuk memanfaatkan pembelajaran mesin secara efektif, kita perlu mengambil langkah mundur dan mempertimbangkan masalahnya secara luas. Pertama, Anda harus memikirkan pertanyaan seperti apa yang ingin Anda jawab. Apakah Anda ingin melakukan analisis eksplorasi dan hanya melihat apakah Anda menemukan sesuatu yang menarik dalam data? Atau apakah Anda sudah memiliki tujuan tertentu dalam pikiran? Seringkali Anda akan memulai dengan tujuan, seperti mendeteksi transaksi pengguna yang curang, membuat rekomendasi film, atau menemukan planet yang tidak dikenal. Jika Anda memiliki tujuan seperti itu, sebelum membangun sistem untuk mencapainya, pertama-tama Anda harus berpikir tentang bagaimana mendefinisikan dan mengukur kesuksesan, dan apa dampak dari solusi yang sukses terhadap keseluruhan bisnis atau tujuan penelitian Anda. Katakanlah tujuan Anda adalah deteksi penipuan.

Kemudian pertanyaan-pertanyaan berikut terbuka:

- Bagaimana cara mengukur apakah prediksi penipuan saya benar-benar berfungsi?
- Apakah saya memiliki data yang tepat untuk mengevaluasi suatu algoritma?
- Jika saya berhasil, apa dampak bisnis dari solusi saya?

Seperti yang telah kita bahas di Bab 5, yang terbaik adalah jika Anda dapat mengukur kinerja algoritma Anda secara langsung menggunakan metrik bisnis, seperti peningkatan laba atau penurunan kerugian. Ini sering sulit dilakukan, meskipun. Pertanyaan yang lebih mudah dijawab adalah “Bagaimana jika saya membuat model yang sempurna?” Jika mendeteksi penipuan dengan sempurna akan menghemat \$100 per bulan perusahaan Anda, kemungkinan penghematan ini mungkin tidak akan cukup untuk menjamin upaya Anda bahkan untuk mulai mengembangkan algoritma. Di sisi lain, jika model tersebut dapat menghemat puluhan ribu dolar perusahaan Anda setiap bulan, masalahnya mungkin perlu ditelusuri.

Katakanlah Anda telah mendefinisikan masalah untuk dipecahkan, Anda tahu sebuah solusi mungkin memiliki dampak yang signifikan untuk proyek Anda, dan Anda telah memastikan bahwa Anda memiliki informasi yang tepat untuk mengevaluasi keberhasilan. Langkah selanjutnya biasanya memperoleh data dan membangun prototipe kerja. Dalam buku ini kita telah berbicara tentang banyak model yang dapat Anda gunakan, dan bagaimana mengevaluasi dan menyesuaikan model-model ini dengan benar. Saat mencoba model, perlu diingat bahwa ini hanya sebagian kecil dari alur kerja ilmu data yang lebih besar, dan pembuatan model sering kali merupakan bagian dari lingkaran umpan balik untuk mengumpulkan data baru, membersihkan data, membangun model, dan menganalisis model. Menganalisis kesalahan yang dibuat model sering kali dapat memberikan informasi tentang apa yang hilang dalam data, data tambahan apa yang dapat dikumpulkan, atau bagaimana tugas dapat dirumuskan ulang untuk membuat pembelajaran mesin lebih efektif. Mengumpulkan lebih banyak atau data yang berbeda atau mengubah sedikit perumusan tugas mungkin memberikan hasil yang jauh lebih tinggi daripada menjalankan pencarian grid tanpa akhir untuk menyesuaikan parameter.

### **Manusia dalam Lingkaran**

Anda juga harus mempertimbangkan apakah dan bagaimana Anda harus memiliki manusia dalam lingkaran. Beberapa proses (seperti deteksi pejalan kaki di mobil yang mengemudi sendiri) perlu segera diambil keputusan. Orang lain mungkin tidak memerlukan tanggapan segera, sehingga memungkinkan manusia untuk mengkonfirmasi keputusan yang tidak pasti. Aplikasi medis, misalnya, mungkin memerlukan tingkat presisi yang sangat tinggi yang mungkin tidak dapat dicapai dengan algoritma pembelajaran mesin saja. Tetapi jika suatu algoritme dapat membuat 90 persen, 50 persen, atau bahkan mungkin hanya 10 persen dari keputusan secara otomatis, itu mungkin sudah meningkatkan waktu respons atau mengurangi biaya. Banyak aplikasi yang didominasi oleh "kasus sederhana," yang algoritma dapat membuat keputusan, dengan relatif sedikit "kasus rumit," yang dapat dialihkan ke manusia.

## **8.2 DARI PROTOTIPE KE PRODUKSI**

Alat yang telah kita bahas dalam buku ini sangat bagus untuk banyak aplikasi pembelajaran mesin, dan memungkinkan analisis dan pembuatan prototipe yang sangat cepat. Python dan scikit-learn juga digunakan dalam sistem produksi di banyak organisasi—bahkan yang sangat besar seperti bank internasional dan perusahaan media sosial global. Namun, banyak perusahaan memiliki infrastruktur yang kompleks, dan tidak selalu mudah untuk memasukkan Python ke dalam sistem ini. Itu belum tentu menjadi masalah. Di banyak perusahaan, tim analitik data bekerja dengan bahasa seperti Python dan R yang memungkinkan pengujian ide dengan cepat, sementara tim produksi bekerja dengan bahasa seperti Go, Scala, C++, dan Java untuk membangun sistem yang tangguh dan skalabel. Analisis data memiliki persyaratan yang berbeda dari membangun layanan langsung, sehingga menggunakan bahasa yang berbeda untuk tugas ini masuk akal. Solusi yang relatif umum adalah mengimplementasikan kembali solusi yang ditemukan oleh tim analitik di dalam kerangka kerja yang lebih besar, menggunakan bahasa berkinerja tinggi. Ini bisa lebih mudah

daripada menyematkan seluruh pustaka atau bahasa pemrograman dan mengonversi dari dan ke format data yang berbeda.

Terlepas dari apakah Anda dapat menggunakan scikit-learn dalam sistem produksi atau tidak, penting untuk diingat bahwa sistem produksi memiliki persyaratan yang berbeda dari skrip analisis satu kali. Jika suatu algoritma digunakan ke dalam sistem yang lebih besar, aspek rekayasa perangkat lunak seperti keandalan, prediktabilitas, runtime, dan persyaratan memori mendapatkan relevansi. Kesederhanaan adalah kunci dalam menyediakan sistem pembelajaran mesin yang berkinerja baik di area ini. Periksa secara kritis setiap bagian dari pemrosesan data dan alur prediksi Anda dan tanyakan pada diri Anda sendiri seberapa besar kerumitan yang dibuat oleh setiap langkah, seberapa kuat setiap komponen terhadap perubahan dalam data atau infrastruktur komputasi, dan apakah manfaat dari setiap komponen menjamin kompleksitas tersebut. Jika Anda sedang membangun sistem pembelajaran mesin yang terlibat, kami sangat menyarankan untuk membaca makalah "Pembelajaran Mesin: Kartu Kredit Berbunga Tinggi dari Hutang Teknis", yang diterbitkan oleh para peneliti di tim pembelajaran mesin Google. Makalah ini menyoroti trade-off dalam menciptakan dan memelihara perangkat lunak pembelajaran mesin dalam produksi dalam skala besar. Sementara masalah utang teknis sangat mendesak dalam proyek skala besar dan jangka panjang, pelajaran yang didapat dapat membantu kita membangun perangkat lunak yang lebih baik bahkan untuk sistem yang berumur pendek dan lebih kecil.

### **8.3 MENGUJI SISTEM PRODUKSI**

Dalam buku ini, kami membahas cara mengevaluasi prediksi algoritmik berdasarkan set tes yang kami kumpulkan sebelumnya. Ini dikenal sebagai evaluasi offline. Jika sistem pembelajaran mesin Anda menghadap ke pengguna, ini hanya langkah pertama dalam mengevaluasi suatu algoritme. Langkah selanjutnya biasanya pengujian online atau pengujian langsung, di mana konsekuensi dari penggunaan algoritma dalam keseluruhan sistem dievaluasi. Mengubah rekomendasi atau hasil pencarian yang ditunjukkan oleh situs web kepada pengguna dapat mengubah perilaku mereka secara drastis dan menyebabkan konsekuensi yang tidak terduga. Untuk melindungi dari kejutan ini, sebagian besar layanan yang dihadapi pengguna menggunakan pengujian A/B, suatu bentuk studi pengguna buta. Dalam pengujian A/B, tanpa sepengertahuan mereka, sebagian pengguna yang dipilih akan diberikan situs web atau layanan menggunakan algoritme A, sedangkan pengguna lainnya akan diberikan algoritme B. Untuk kedua grup, metrik keberhasilan yang relevan akan dicatat untuk periode waktu yang ditentukan. Kemudian, metrik dari algoritma A dan algoritma B akan dibandingkan, dan pemilihan antara dua pendekatan akan dibuat berdasarkan metrik tersebut. Menggunakan pengujian A/B memungkinkan kami mengevaluasi algoritme "di alam liar", yang mungkin membantu kami menemukan konsekuensi yang tidak terduga saat pengguna berinteraksi dengan model kami. Seringkali A adalah model baru, sedangkan B adalah sistem yang sudah mapan. Ada mekanisme yang lebih rumit untuk pengujian online yang melampaui pengujian A/B, seperti algoritma bandit. Pengantar yang bagus untuk subjek ini dapat ditemukan dalam buku *Bandit Algorithms for Website Optimization* oleh John Myles White (O'Reilly).

## 8.4 MEMBUAT PENGUKUR SENDIRI

Buku ini telah membahas berbagai alat dan algoritma yang diimplementasikan dalam scikit-belajar yang dapat digunakan pada berbagai tugas. Namun, seringkali akan ada beberapa pemrosesan tertentu yang perlu Anda lakukan untuk data Anda yang tidak diterapkan di scikit-learn. Mungkin cukup dengan melakukan praproses data Anda sebelum meneruskannya ke model atau saluran scikit-learn Anda. Namun, jika prapemrosesan Anda bergantung pada data, dan Anda ingin menerapkan pencarian grid atau validasi silang, segalanya menjadi lebih rumit.

Dalam Bab 6 kita membahas pentingnya menempatkan semua pemrosesan yang bergantung pada data di dalam loop validasi silang. Jadi bagaimana Anda bisa menggunakan pemrosesan Anda sendiri bersama dengan alat scikit-learn? Ada solusi sederhana: buat estimator Anda sendiri! Menerapkan estimator yang kompatibel dengan antarmuka scikit-learn, sehingga dapat digunakan dengan Pipeline, GridSearchCV, dan cross\_val\_score, cukup mudah. Anda dapat menemukan instruksi terperinci dalam dokumentasi scikit-learn, tetapi inilah intinya. Cara paling sederhana untuk mengimplementasikan kelas transformer adalah dengan mewarisi dari BaseEstimator dan TransformerMixin, lalu mengimplementasikan init , fit, dan predict fungsi seperti ini:

In[1]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class MyTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, first_parameter=1, second_parameter=2):
        # All parameters must be specified in the __init__ function
        self.first_parameter = 1
        self.second_parameter = 2

    def fit(self, X, y=None):
        # fit should only take X and y as parameters
        # Even if your model is unsupervised, you need to accept a y argument!

        # Model fitting code goes here
        print("fitting the model right here")
        # fit returns self
        return self

    def transform(self, X):
        # transform takes as parameter only X

        # Apply some transformation to X
        X_transformed = X + 1
        return X_transformed
```

Menerapkan classifier atau regressor bekerja dengan cara yang sama, hanya sebagai ganti Transformer Mixin Anda perlu mewarisi dari ClassifierMixin atau RegressorMixin. Selain itu, alih-alih menerapkan transformasi, Anda akan menerapkan prediksi.

Seperti yang dapat Anda lihat dari contoh yang diberikan di sini, mengimplementasikan estimator Anda sendiri membutuhkan kode yang sangat sedikit, dan sebagian besar pengguna scikit-learn membangun koleksi model khusus dari waktu ke waktu.

### **Ke mana harus pergi dari sini?**

Buku ini memberikan pengantar pembelajaran mesin dan akan menjadikan Anda seorang praktisi yang efektif. Namun, jika Anda ingin meningkatkan keterampilan pembelajaran mesin Anda, berikut adalah beberapa saran buku dan sumber daya yang lebih khusus untuk diselidiki untuk menyelam lebih dalam.

### **8.5 KERANGKA DAN PAKET PEMBELAJARAN MESIN LAINNYA**

Meskipun scikit-learn adalah paket favorit kami untuk pembelajaran mesin1 dan Python adalah bahasa favorit kami untuk pembelajaran mesin, ada banyak pilihan lain di luar sana. Bergantung pada kebutuhan Anda, Python dan scikit-learn mungkin bukan yang paling cocok untuk situasi khusus Anda. Seringkali menggunakan Python sangat bagus untuk mencoba dan mengevaluasi model, tetapi layanan web dan aplikasi yang lebih besar lebih sering ditulis dalam Java atau C++, dan pengintegrasian ke dalam sistem ini mungkin diperlukan agar model Anda dapat diterapkan. Alasan lain Anda mungkin ingin melihat melampaui scikit-learn adalah jika Anda lebih tertarik pada pemodelan statistik dan inferensi daripada prediksi. Dalam hal ini, Anda harus mempertimbangkan paket statsmodel untuk Python, yang mengimplementasikan beberapa model linier dengan antarmuka yang lebih berorientasi statistik. Jika Anda belum menikah dengan Python, Anda mungkin juga mempertimbangkan untuk menggunakan R, lingua franca lain dari ilmuwan data. R adalah bahasa yang dirancang khusus untuk analisis statistik dan terkenal dengan kemampuan visualisasi yang sangat baik dan ketersediaan banyak (seringkali sangat khusus) paket pemodelan statistik.

Paket pembelajaran mesin populer lainnya adalah vokal wabbit (sering disebut vw untuk menghindari kemungkinan terpelintir lidah), paket pembelajaran mesin yang sangat dioptimalkan yang ditulis dalam C++ dengan antarmuka baris perintah. vw sangat berguna untuk kumpulan data besar dan untuk streaming data. Untuk menjalankan algoritme pembelajaran mesin yang didistribusikan pada sebuah cluster, salah satu solusi paling populer pada saat penulisan ini adalah mllib, perpustakaan Scala yang dibangun di atas lingkungan komputasi terdistribusi percikan.

### **8.6 PEMERINGKATAN, SISTEM REKOMENDASI, DAN JENIS PEMBELAJARAN LAINNYA**

Karena ini adalah buku pengantar, kami berfokus pada tugas pembelajaran mesin yang paling umum: klasifikasi dan regresi dalam pembelajaran terawasi, dan pengelompokan dan dekomposisi sinyal dalam pembelajaran tanpa pengawasan. Ada lebih banyak jenis pembelajaran mesin di luar sana, dengan banyak aplikasi penting. Ada dua topik yang sangat penting yang tidak kami bahas dalam buku ini. Yang pertama adalah peringkat, di mana kami ingin mengambil jawaban atas kueri tertentu, diurutkan berdasarkan relevansinya. Anda mungkin sudah menggunakan sistem peringkat hari ini; ini adalah bagaimana mesin pencari beroperasi. Anda memasukkan kueri penelusuran dan memperoleh daftar jawaban yang diurutkan, diberi peringkat berdasarkan relevansinya. Pengantar yang bagus untuk peringkat disediakan dalam buku Manning, Raghavan, dan Schütze, Pengantar Pengambilan Informasi.

Topik kedua adalah sistem rekomendasi, yang memberikan saran kepada pengguna berdasarkan preferensi mereka. Anda mungkin pernah menemukan sistem pemberi rekomendasi di bawah judul seperti ‐Orang yang Mungkin Anda Kenal‐, ‐Pelanggan yang Membeli Barang Ini Juga Membeli‐, atau ‐Pilihan Teratas untuk Anda‐. Ada banyak literatur tentang topik ini, dan jika Anda ingin terjun langsung ke dalamnya, Anda mungkin tertarik dengan ‐tantangan hadiah Netflix‐ yang sekarang klasik, di mana situs streaming video Netflix merilis kumpulan data preferensi film yang besar dan menawarkan hadiah sebesar \$1 juta kepada tim yang dapat memberikan rekomendasi terbaik. Aplikasi umum lainnya adalah prediksi deret waktu (seperti harga saham), yang juga memiliki banyak literatur yang dikhususkan untuk itu. Ada lebih banyak tugas pembelajaran mesin di luar sana—lebih banyak dari yang dapat kami sebutkan di sini—and kami mendorong Anda untuk mencari informasi dari buku, makalah penelitian, dan komunitas online untuk menemukan paradigma yang paling sesuai dengan situasi Anda.

### **8.7 PEMODELAN PROBABILISTIK, INFERENSI, DAN PEMROGRAMAN PROBABILISTIK**

Sebagian besar paket pembelajaran mesin menyediakan model pembelajaran mesin yang telah ditentukan sebelumnya yang menerapkan satu algoritme tertentu. Namun, banyak masalah dunia nyata memiliki struktur tertentu yang, ketika dimasukkan dengan benar ke dalam model, dapat menghasilkan prediksi dengan kinerja yang jauh lebih baik. Seringkali, struktur masalah tertentu dapat diekspresikan dengan menggunakan bahasa teori probabilitas. Struktur seperti itu biasanya muncul dari memiliki model matematis dari situasi yang ingin Anda prediksi. Untuk memahami apa yang dimaksud dengan masalah terstruktur, perhatikan contoh berikut.

Katakanlah Anda ingin membangun aplikasi seluler yang menyediakan perkiraan posisi yang sangat rinci di ruang terbuka, untuk membantu pengguna menavigasi situs bersejarah. Ponsel menyediakan banyak sensor untuk membantu Anda mendapatkan pengukuran lokasi yang tepat, seperti GPS, akselerometer, dan kompas. Anda juga memiliki peta area yang tepat. Masalah ini sangat terstruktur. Anda tahu di mana jalur dan tempat menarik dari peta Anda. Anda juga memiliki posisi kasar dari GPS, dan akselerometer serta kompas di perangkat pengguna memberi Anda pengukuran relatif yang sangat tepat.

Tetapi menggabungkan semua ini ke dalam sistem pembelajaran mesin kotak hitam untuk memprediksi posisi mungkin bukan ide terbaik. Ini akan membuang semua informasi yang sudah Anda ketahui tentang cara kerja dunia nyata. Jika kompas dan akselerometer memberi tahu Anda bahwa pengguna pergi ke utara, dan GPS memberi tahu Anda bahwa pengguna pergi ke selatan, Anda mungkin tidak dapat mempercayai GPS. Jika perkiraan posisi Anda memberi tahu Anda bahwa pengguna baru saja melewati tembok, Anda juga harus sangat skeptis. Dimungkinkan untuk mengekspresikan situasi ini menggunakan model probabilistik, dan kemudian menggunakan pembelajaran mesin atau inferensi probabilistik untuk mengetahui seberapa besar Anda harus memercayai setiap pengukuran, dan untuk mempertimbangkan tebakan terbaik untuk lokasi pengguna.

Setelah Anda menyatakan situasi dan model Anda tentang bagaimana berbagai faktor bekerja sama dengan cara yang benar, ada metode untuk menghitung prediksi menggunakan model khusus ini secara langsung. Yang paling umum dari metode ini disebut bahasa pemrograman probabilistik, dan mereka menyediakan cara yang sangat elegan dan ringkas untuk mengekspresikan masalah pembelajaran. Contoh bahasa pemrograman probabilistik yang populer adalah PyMC (yang dapat digunakan dengan Python) dan Stan (framework yang dapat digunakan dari beberapa bahasa, termasuk Python). Sementara paket-paket ini memerlukan beberapa pemahaman tentang teori probabilitas, mereka menyederhanakan pembuatan model baru secara signifikan.

### **8.8 JARINGAN SARAF**

Sementara kami membahas masalah jaringan saraf secara singkat di Bab 2 dan 7, ini adalah area pembelajaran mesin yang berkembang pesat, dengan inovasi dan aplikasi baru diumumkan setiap minggu. Terobosan terbaru dalam pembelajaran mesin dan kecerdasan buatan, seperti kemenangan program Alpha Go melawan juara manusia dalam game Go, kinerja pemahaman ucapan yang terus meningkat, dan ketersediaan terjemahan ucapan yang hampir seketika, semuanya didorong oleh kemajuan ini. Sementara kemajuan di bidang ini begitu cepat sehingga setiap referensi terkini tentang keadaan seni akan segera ketinggalan zaman, buku terbaru Deep Learning oleh Ian Goodfellow, Yoshua Bengio, dan Aaron Courville (MIT Press) adalah pengantar komprehensif ke subjek.

### **8.9 MENSKALAKAN KE KUMPULAN DATA YANG LEBIH BESAR**

Dalam buku ini, kami selalu berasumsi bahwa data yang kami kerjakan dapat disimpan dalam array NumPy atau matriks sparse SciPy di memori (RAM). Meskipun server modern sering kali memiliki ratusan gigabyte (GB) RAM, ini adalah batasan mendasar pada ukuran data yang dapat Anda gunakan. Tidak semua orang mampu membeli mesin sebesar itu, atau bahkan menyewanya dari penyedia cloud. Namun, di sebagian besar aplikasi, data yang digunakan untuk membangun sistem pembelajaran mesin relatif kecil, dan beberapa set data pembelajaran mesin terdiri dari ratusan gigabita data atau lebih. Hal ini membuat perluasan RAM atau menyewa mesin dari penyedia cloud menjadi solusi yang layak dalam banyak kasus. Namun, jika Anda perlu bekerja dengan terabyte data, atau Anda perlu memproses data dalam jumlah besar dengan anggaran terbatas, ada dua strategi dasar: pembelajaran di luar inti dan paralelisasi melalui klaster.

Pembelajaran *out-of-core* menggambarkan pembelajaran dari data yang tidak dapat disimpan dalam memori utama, tetapi pembelajaran terjadi pada satu komputer (atau bahkan satu prosesor di dalam komputer). Data dibaca dari sumber seperti hard disk atau jaringan baik satu sampel pada satu waktu atau dalam potongan beberapa sampel, sehingga setiap potongan cocok dengan RAM. Subset data ini kemudian diproses dan model diperbarui untuk mencerminkan apa yang dipelajari dari data. Kemudian, potongan data ini dibuang dan bit data berikutnya dibaca. Pembelajaran di luar inti diterapkan untuk beberapa model di scikit-learn, dan Anda dapat menemukan detailnya di panduan pengguna online. Karena pembelajaran di luar inti memerlukan semua data untuk diproses oleh satu komputer, hal ini

dapat menyebabkan runtime yang lama pada kumpulan data yang sangat besar. Juga, tidak semua algoritma pembelajaran mesin dapat diimplementasikan dengan cara ini.

Strategi lain untuk penskalaan adalah mendistribusikan data melalui beberapa mesin dalam cluster komputasi, dan membiarkan setiap komputer memproses sebagian data. Ini bisa jauh lebih cepat untuk beberapa model, dan ukuran data yang dapat diproses hanya dibatasi oleh ukuran cluster. Namun, perhitungan seperti itu seringkali membutuhkan infrastruktur yang relatif kompleks. Salah satu platform komputasi terdistribusi paling populer saat ini adalah platform percikan yang dibangun di atas Hadoop. spark menyertakan beberapa fungsi pembelajaran mesin dalam paket MLLib. Jika data Anda sudah ada di sistem file Hadoop, atau Anda sudah menggunakan spark untuk memproses data sebelumnya, ini mungkin opsi termudah. Namun, jika Anda belum memiliki infrastruktur seperti itu, membangun dan mengintegrasikan spark cluster mungkin merupakan upaya yang terlalu besar. Paket vw yang disebutkan sebelumnya menyediakan beberapa fitur terdistribusi dan mungkin menjadi solusi yang lebih baik dalam kasus ini.

## **8.10 MENGASAH KETERAMPILAN ANDA**

Seperti halnya banyak hal dalam hidup, hanya latihan yang akan memungkinkan Anda menjadi ahli dalam topik yang kita bahas dalam buku ini. Ekstraksi fitur, prapemrosesan, visualisasi, dan pembuatan model dapat sangat bervariasi antara tugas yang berbeda dan kumpulan data yang berbeda. Mungkin Anda cukup beruntung karena sudah memiliki akses ke berbagai kumpulan data dan tugas. Jika Anda belum memikirkan tugas, tempat yang baik untuk memulai adalah kompetisi pembelajaran mesin, di mana kumpulan data dengan tugas yang diberikan diterbitkan, dan tim bersaing dalam menciptakan prediksi terbaik. Banyak perusahaan, organisasi nirlaba, dan universitas menyelenggarakan kompetisi ini. Salah satu tempat paling populer untuk menemukannya adalah Kaggle, sebuah situs web yang secara rutin mengadakan kompetisi ilmu data, beberapa di antaranya memiliki hadiah uang yang cukup besar.

Forum Kaggle juga merupakan sumber informasi yang baik tentang alat dan trik terbaru dalam pembelajaran mesin, dan berbagai kumpulan data tersedia di situs. Bahkan lebih banyak set data dengan tugas terkait dapat ditemukan di platform OpenML, yang menampung lebih dari 20.000 set data dengan lebih dari 50.000 tugas pembelajaran mesin terkait. Bekerja dengan kumpulan data ini dapat memberikan peluang besar untuk melatih keterampilan pembelajaran mesin Anda. Kerugian dari kompetisi adalah mereka sudah menyediakan metrik tertentu untuk dioptimalkan, dan biasanya set data yang telah diproses sebelumnya. Ingatlah bahwa mendefinisikan masalah dan mengumpulkan data juga merupakan aspek penting dari masalah dunia nyata, dan bahwa merepresentasikan masalah dengan cara yang benar mungkin jauh lebih penting daripada memeras persentase akurasi terakhir dari pengklasifikasi.

- Alain, G., Bengio, Y., Yao, L., Éric Thibodeau-Laufer, Yosinski, J., and Vincent, P. (2015). GSNs: Generative stochastic networks. arXiv:1503.05571.
- Bachman, P. and Precup, D. (2015). *Variational generative stochastic networks with collaborative shaping*. In Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015 , pages 1964–1972.
- Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015b). Scheduled sampling for sequence prediction with recurrent neural networks. Technical report, arXiv:1506.03099.
- Bengio, Y. (2015). *Early inference in energy-based models approximates back-propagation*. Technical Report arXiv:1510.02777, Universite de Montreal.
- Bornschein, J. and Bengio, Y. (2015). Reweighted wake-sleep. In ICLR'2015.
- Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. (2015). *MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems*.
- Chrupala, G., Kadar, A., and Alishahi, A. (2015). *Learning language through pictures*.
- Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Low precision arithmetic for deep learning. In Arxiv:1412.7024, ICLR'2015 Workshop.
- Denton, E., Chintala, S., Szlam, A., and Fergus, R. (2015). *Deep generative image models using a Laplacian pyramid of adversarial networks*. NIPS.
- Desjardins, G., Simonyan, K., Pascanu, R., (2015). *Natural neural networks*. In et al. Advances in Neural Information Processing Systems, pages 2062–2070.
- Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2015). Learning to generate chairs with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1538–1546.
- Fang, H., Gupta, S., Iandola, F., Srivastava, R., Deng, L., Dollár, P., Gao, J., He, X., Mitchell, M., Platt, J. C., Zitnick, C. L., and Zweig, G. (2015). *From captions to visual concepts and back*.
- Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. (2015). *Learning visual feature spaces for robotic manipulation with deep spatial autoencoders*.
- Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with Bernoulli approximate variational inference.
- Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2015). *Region-based convolutional networks for accurate object detection and segmentation*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: *Surpassing human-level performance on ImageNet classification*.
- Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2014). On using very large target vocabulary for neural machine translation.
- Joulin, A. and Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets.
- Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). *Grid long short-term memory*.

- Kamyshanska, H. and Memisevic, R. (2015). The potential energy of an autoencoder. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Karpathy, A. and Li, F.-F. (2015). Deep visual-semantic alignments for generating image descriptions. In CVPR'2015.
- Kumar, A., Irsoy, O., Su, J., Bradbury, J., English, R., Pierce, B., Ondruska, P., Iyyer, M., Gulrajani, I., and Socher, R. (2015). *Ask me anything: Dynamic memory networks for natural language processing*.
- Li, Y., Swersky, K., and Zemel, R. S. (2015). Generative moment matching networks.
- Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). *Learning entity and relation embeddings for knowledge graph completion*. In Proc. AAAI'15 .
- MacLaurin, D., Duvenaud, D., and Adams, R. P. (2015). *Gradient-based hyperparameter optimization through reversible learning*.
- Pinheiro, P. H. O. and Collobert, R. (2015). *From image-level to pixel-level labeling with convolutional networks*. In Conference on Computer Vision and Pattern Recognition (CVPR).
- Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks.
- Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). *Deep unsupervised learning using nonequilibrium thermodynamics*.
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision.
- Vincent, P., de Brébisson, A., and Bouthillier, X. (2015). Efficient exact gradient update for training deep networks with very large sparse targets. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28. pages 1108–1116. Curran Associates, Inc.
- Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. (2015). *Deep image: Scaling up image recognition*.
- Zaremba, W. and Sutskever, I. (2015). Reinforcement learning neural Turing machines.